Bachelor thesis
Computing Science

Radboud University

Parsons' puzzles as formative assessment

Author:
Kirsten Kingma
s4449037

First supervisor:
Dr. Maria Kallia
maria.kallia@ru.nl

Second supervisor/assessor:
Prof. Dr. Erik Barendsen
erik.barendsen@ru.nl

January 15, 2020

Abstract

Formative assessment provides continuous feedback in order to monitor and improving students' learning. A tool to assess programming skills which shows great potential is Parsons' puzzles. This study explores students' and teachers' perspectives to investigate the potential of Parsons' puzzles as a tool to generate formative feedback. We conducted a study on a class of students in secondary education where we split the class in two. One group participated in Parson's puzzles and the other group participated in writing tasks as formative assessment. After three weeks of experimentation the analysis of the results indicates that Parsons' puzzles take less time to correct than writing tasks, but are not challenging enough for students with a high skill level in programming. Students with a low skill level could benefit from Parsons' puzzles to enhance their understanding in programming, because of the lower difficulty. Thus we conclude that when a teacher is mindful of the difference in difficulty of Parsons' puzzles and writing task, and matches these accordingly to the skill level of their students, Parsons' puzzles can be a useful tool to generate effective formative assessment.

Contents

Chapter 1

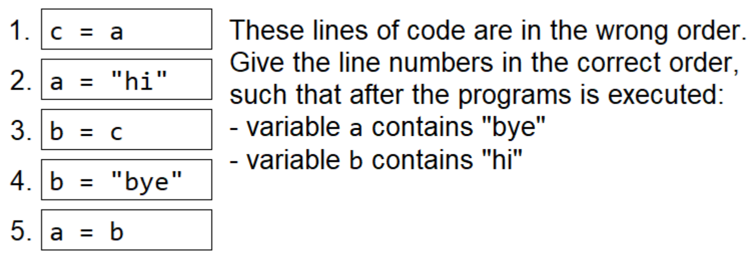
Introduction

In every subject at secondary education, getting and giving feedback on a student's skill is an important part of the education. In a big meta-research about the effect of different influences on learning (Hattie & Timperley, 2007), feedback scored first place. But how does a teacher give good feedback? For a long time teachers have practiced summative assessment: giving grades on the final product a student produced. This way, students get structured feedback on finished work. But like a cook tasting their food while they are making it, students could profit from getting feedback during their learning. This method is known as formative assessment and its basic aim is to provide continuous feedback for monitoring and improving students' learning.

A meta-analysis (607 effect sizes; 23,663 observations) performed in 1998 suggested that feedback interventions in any context improved performance with an average effect size of .41 (Kluger & DeNisi, 1996). Since then, multiple studies and articles argued that formative assessment is essential in the classroom(Black & Wiliam, 2009, 1998; Nicol & Macfarlane-Dick, 2006).

Formative assessment can be implemented in many different ways, but for computer science there is still a lot to be done, unfortunately. When teachers assess programming skills of students, the most used assignment is to write code. Although writing code is essential and necessary for both students' practice and understanding, it can be time consuming for a student and for a teacher to grade. It may not even be the best option when students start learning new programming concepts or constructs.

A potential solution to these problems could be Parsons' puzzles (Parsons & Haden, 2006) (more information is provided in related work at section 2.2). In figure 1.1 we can see an example of a Parsons' puzzle testing a basic programming concept in the programming language Python.



```
1. c = a
2. a = "hi"
3. b = c
4. b = "bye"
5. a = b
```

These lines of code are in the wrong order.
Give the line numbers in the correct order,
such that after the programs is executed:
- variable a contains "bye"
- variable b contains "hi"

Figure 1.1: Example of a Parsons" Puzzle

Parsons' puzzles are a tool to assess programming skills in an entertaining puzzle-like format. They maximize engagement and since each puzzle solution is a complete sample of well-written code, use of the tool exposes students to good programming practice (Parsons & Haden, 2006). Research has shown that not only do Parsons' take less time to grade, they reduce the cognitive load of the assignment significantly (Ericson, Margulieux, & Rick, 2017). This is particularly important since previous research suggests that exercises with a high cognitive load can slow gaining expertise (Sweller, 1988).

The aim of this research is to investigate the potential of Parsons' puzzles as a tool for formative assessment. Therefore, our research questions are formulated as follows:

- Can Parsons' puzzles be used as a tool to generate effective formative assessment?

To help us with answering this question, we defined two sub-questions:

- Are Parsons' puzzles beneficial for teachers as a formative assessment?
- Are Parsons' puzzles beneficial for students to enhance their understanding in programming?

1.1 Reflection

The focal reason that I chose Parsons' puzzles as formative assessment as the subject of my thesis is that as a teacher of computer science in secondary education, I have encountered difficulties with assessing my students' knowledge. During teaching lessons, I encountered some difficulties with assessing the skill of students. The most obvious way of assessing the students' skill level in programming is asking them to write code. But, from my personal experience as a teacher, this method alone isn't efficient.

Firstly, if the students write their answers on paper, a lot of crucial little (syntax) errors are made which would have never happened when the student was writing in an IDE.

Secondly, the cognitive load of these writing assignments. When writing code, looking up parts of the solution and small pieces of example code is essential. When I instruct students in an assignment, I always set a ground rule for asking questions. "*When you encounter difficulties, first ask your neighbour or search for an answer on the internet and if you still have not found the solution after trying those options, you can ask me.*" I believe this prepares them the best for writing code independently. The hints students can get from all these sources, decreases the cognitive load of writing code immensely.

The third problem I encounter with writing assignments is giving feedback. Without proper feedback on their code, my students keep on making the same mistakes over and over. But when students are just starting they often struggle with giving pseudo code to answer the question. Giving helpful feedback is very hard when the code is just completely incomprehensible. Because trying to understand code someone else has written can take a lot of time, even when it is completely correct. Trying to find out the misunderstandings students have from their code, which is filled with errors, is very difficult.

I strive to give my students the best education possible. And assessing their skill level and giving helpful feedback is an essential part in their education. But I believe asking students to simply write code is not an efficient experience for students and teachers alike. That is why I want to assess my students' skill level by including different types of assignments. I have tried asking them to fix code, trace code with a table and summarizing code in their own words. But these methods do not test the application objective of the Bloom taxonomy that writing does (Bloom, Krathwohl, & Masia, 1984).

This thesis gives me the opportunity to review Parsons' puzzles as a tool of assessing the students' skill level in a more efficient way. Learning new methods for assessment is important to me as a young teacher, because it improves my teaching practice and therefore the efficiency of my feedback and grading for students. I believe improving those elements in education can help students to learn more efficiently and most importantly, enjoy learning how to code. This can lower the threshold of learning how to code for many more students to come.

1.2 Overview

In this thesis we first talk about related work and their influence on our research. Then we explain the methodology of our research and present the results we gathered. After that, we summarize our findings and discuss the implications for teaching. Finally, we present the conclusions gathered from our results.

Chapter 2

Related work

In this chapter we present the related work to the different themes we discuss during this thesis. Because the thesis is about Parsons' puzzles as formative assessment, these are the first two themes we discuss. Next, we talk about literature researching exit cards, the think-pair-share exercise and self-efficacy.

2.1 Formative assessment

For our research we adapt these definitions:

> "'Assessment' refers to all those activities undertaken by teachers, and by the students in assessing themselves, which provide information to be used as feedback to modify the teaching and learning activities in which they are engaged. Such assessment become 'formative assessment' when the evidence is actually used to adapt the teaching to meet the needs."
> (Black & Wiliam, 1998, 2)

Thus formative assessment is a method to provide feedback for students. The type of feedback we give has a big influence on it's effectiveness. For example, normative feedback, which relies on teacher comparisons of students, should be avoided, because it tends to motivate students for extrinsic reasons, promotes performance goals, and can lower expectations for success (Cauley & McMillan, 2010). To promote performance goals, feedback from formative assessments should reduce social comparisons and instead emphasize progress toward achieving learning targets (Maehr & Anderman, 1993). That is why we refrain from giving grades or comments which students could compare to each other in this study.

Furthermore, students need to partner with their teacher to continuously monitor their current level of attainment in relation to agreed-upon expectations so they can set goals for what to learn next and thus play a role in managing their own progress (Stiggins, 2005).

So, feedback should not only focus on students' current performance, e.g. in a test, but should also engage teachers and students in a conversation where students' learning progress is being regarded in relation with learning goals and expectations.

2.2 Parsons' puzzles

Practice exercises for programming basics can be very tedious. This makes it difficult to motivate students. In their paper, Parsons and Haden describe Parsons' Programming Puzzles, an automated, interactive tool that provides practice with basic programming principles in an entertaining puzzle-like format (Parsons & Haden, 2006). Solving Parsons problems takes significantly less time than fixing code with errors or than writing the equivalent code. Additionally, as noticed by Ericson et al., there is no statistically significant difference in the learning performance, or in a students' retention of the knowledge one week later (Ericson et al., 2017).

We think Parsons' puzzles are ideal in formative assessment because feedback from a Parsons' puzzle makes clear what students don't know (in both syntax and logic) and it is much less ambiguously than marks from a code writing problem (Denny, Luxton-Reilly, & Simon, 2008). This ensures that a teacher can quickly see what the progress of a student is, and give them appropriate feedback.

There are different versions of Parsons' puzzles. 'Distractors' can be added to the available lines of code. These are lines of code that are not part of the solution. There is no difference in the transfer of task performance whether students made tasks with distractors or without them (Harms, Chen, & Kelleher, 2016). However, the distractors increased learners' cognitive load, decreased their success at completing Parsons' puzzles by 26%, and increased learners' time on a task by 14% (Harms et al., 2016).

One could also make the Parsons' puzzles 'two-dimensional'. The vertical dimension is used to order the lines, whereas the horizontal dimension is used to change control flow and code blocks based on indentation as in Python. Two-dimensional puzzles are more complicated when compared to similar puzzles where only the order of the lines needs to be solved (Ihantola & Karavirta, 2011).

A teacher can give line-based or execution based feedback on these Parsons' puzzles. Line-based feedback, is feedback in terms of the order and indentation of the expected correct solution by highlighting code fragments that need to be moved somehow in order to fix the program. Execution based feedback checks the expected results of the code.

In their study, Helminen, Ihantola, Karavirta, and Alaoutinen found that feedback based on execution, resulted in programs which were more frequently executable when feedback was requested and, overall, feedback was requested less frequently, as opposed to line-based feedback (Helminen et al., 2013). At first we did not let the students make the tasks in an IDE in our study, but on paper. Later in the study we let the students fill in their answers in an online environment where students can not execute their code. This made execution based feedback impossible, but we find it more important that the teacher can have a conversation with the student when they give feedback.

Morrison, Margulieux, Ericson, and Guzdial suggested to add sub-goal labels to Parsons' puzzles. This refers to a strategy that helps students to deconstruct problem solving procedures into sub-goals, functional parts of the overall procedure, to better recognize the fundamental components of the problem solving process (Atkinson, Catrambone, & Merrill, 2003). In their study, students who were given sub-goal labels performed statistically better than the groups that did not receive sub-goal labels or were asked to generate sub-goal labels (Morrison et al., 2016). The authors also concluded that a low cognitive load assessment, Parsons' puzzles, can be more sensitive to student learning gains than traditional code generation problems.
In our experiment, the problems presented to the students are very small, so giving sub-goals was unfortunately not an option.

2.3 Exit cards

Exit cards are formative evaluations of students' knowledge undertaken at every class meeting. If students want to exit the classroom at the end of a lesson, they need to complete an exit-card. An exit card can consist of a reflection. In their study, Patka, Wallin-Ruschman, Wallace, and Robbins asked students the following questions:

1. What did you learn?
2. What are you confused about?
3. What hindered your learning today?
4. What helped your learning today?

Students hand wrote their reflection on a sheet of paper or index card which was submitted to the instructor at the end of class (Patka et al., 2016). Unfortunately, 40.68% of exit cards included ‘nothing’ for at least one response. Students may not be taking the Exit Cards seriously and/or were not incentivized to provide detailed answers. These type of exit Cards are often called: ”One Minute Papers”.

We want to give students more incentive to answer the exit card. So we put Parsons’ puzzles and writing tasks on them. This makes reflection no longer the purpose of exit cards. In this form, they can be used as a quick assessment of a student’s understanding of a particular concept or skill (Fattig & Taylor, 2007). This makes exit cards ideal for formative assessment.

2.4 Think-pair-share

Think-Pair-Share is a cooperative discussion strategy that was first developed by Professor Frank Lyman and his colleagues at the University of Maryland in 1981. This teaching-learning strategy works in three phases:

1. Think. The teacher provokes students’ thinking with a question. The students should take a few minutes to think about the question.

2. Pair. Students pair up to talk about the answers they came up with. They can compare their mental or written notes and identify the answers they think are best, most convincing, or most unique.

3. Share. After students talk in pairs for a few minutes, the teacher calls for pairs to share their thinking with the rest of the class.

(Robertson, 2006)

Because students are given time to think about their own responses before they discuss it with their peers, this strategy helps students to form their own opinion. After they have thought about their own response, they get the chance to communicate with one classmate to discuss their answers. This gives the students a possibility to change their answers or to gain new conclusions. Then, when they share their answers with the rest of the group, everybody has the chance to voice their opinion about the given answers.

In a study with a small sample size evidence was found that think-pair-share is effective in promoting critical thinking in students (Kaddoura, 2013). Because the quality of the feedback from the students in our experiment can be enhanced if they think more critically, think-pair-share seems the best strategy to use. It exhausts all the possible opinions the different students could have.

2.5 Self-efficacy

Perceived self-efficacy is concerned with people's beliefs in their ability to influence events that affect their lives. This core belief is the foundation of human motivation, performance accomplishments, and emotional well-being (Bandura, 2006). As van Dinther, Dochy, and Segers put it:

> We certainly are convinced after reading all the studies and the presented evidence that self-efficacy is vital to academic performance and that self-efficacy of students can be affected positively. (van Dinther et al., p.105, 2011)

That is why we study the difference in self-efficacy of students after they have performed the different tasks.

Chapter 3

Methodology

To research if Parsons' puzzles can be used to generate effective formative assessment, we did a study on a class of students in secondary education.

3.1 Philosophical assumptions

The basic paradigm underpinning our research is the interpretivism paradigm. The basic reason for this is that our research is focused on students' experiences with different assessment tools as well as the teacher's reflective experience. We acknowledge that each student may perceive, experience and assign meaning to the assessment process in different ways. In this research study, we are interested in capturing both the different and shared way in which students experience the same phenomenon. The opposite paradigm, positivism, states that understanding of phenomena in reality must be measured and supported by evidence. While the root of interpretivism is that humans interpret their world and then act based on their meaning making and their interpretation. So methods that can be used should be consistent with human and social sciences (Hammersley, 2012).

Using interpretivism, a single phenomenon may have multiple interpretations rather than a truth that can be determined by a process of measurement (Pham, 2018). The advantages of interpretivism include that we can describe certain objects or events and understand them in social context. We get an insider's insight in our research objects, which provides us with even more authentic information (Tuli, 2010). This is perfect for our research, because the students' perspective of Parsons' puzzles is, determines the answer of our sub-question: 'Are Parsons' puzzles beneficial for students to enhance their understanding in programming?'

Unfortunately, interpretivism research tends to be very subjective. We discuss this further in the limitations of the study in section 3.3.

3.2 Methodological approach

Our research methodology is mainly qualitative in nature. This is in line with the interpretivism paradigm as the focus of qualitative research is to provide insights in the meanings, experiences and perspectives of the participants.

3.2.1 Participants

We conducted our research on a class of 16 students in their final year of secondary education in the Netherlands. They follow the VWO (university preparatory education) stream, which is a six-year education stream with a focus on theoretical knowledge, that prepares students to follow a bachelor's degree (WO) at a research university. The students have an average age of 18. 15 of the students are male and 1 is female.

Each student provided consent with a consent form, as you can see in ethical considerations in section 3.3. All students participated because the exercises are a part of their curriculum. Prior to the study, they all followed a computing science course for at least two years. Every week the class had two lessons of one hour with another lesson or a break of one hour in between.

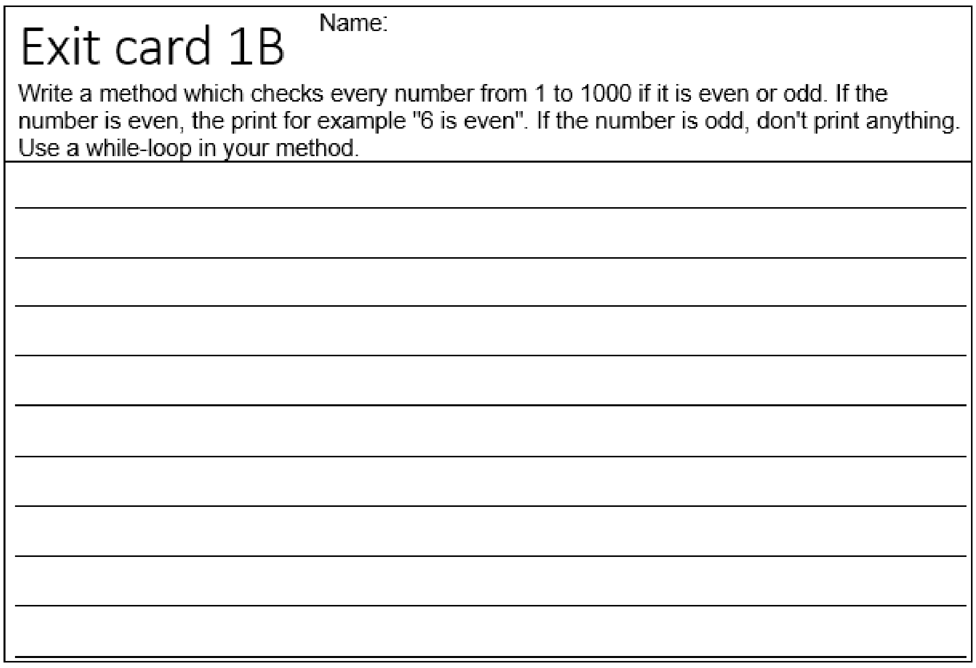
3.2.2 Data collection method

The data were collected from three phases in our study:

1. Before the feedback sessions. A pre-test to create equivalent groups.
2. Three sessions where students get formative feedback. A self-efficacy questionnaire taken after the third feedback session. A log by the teacher.
3. After the feedback sessions. A think-pair-share session about students' perspectives.

The pre-test was done in a controlled classroom with all participants attending at the same time. The test was given on paper. The results were analyzed and we used them to assign each student to one of two groups. The goal of the pre-test was to create two groups of students, were in each group, different skill-levels are present.

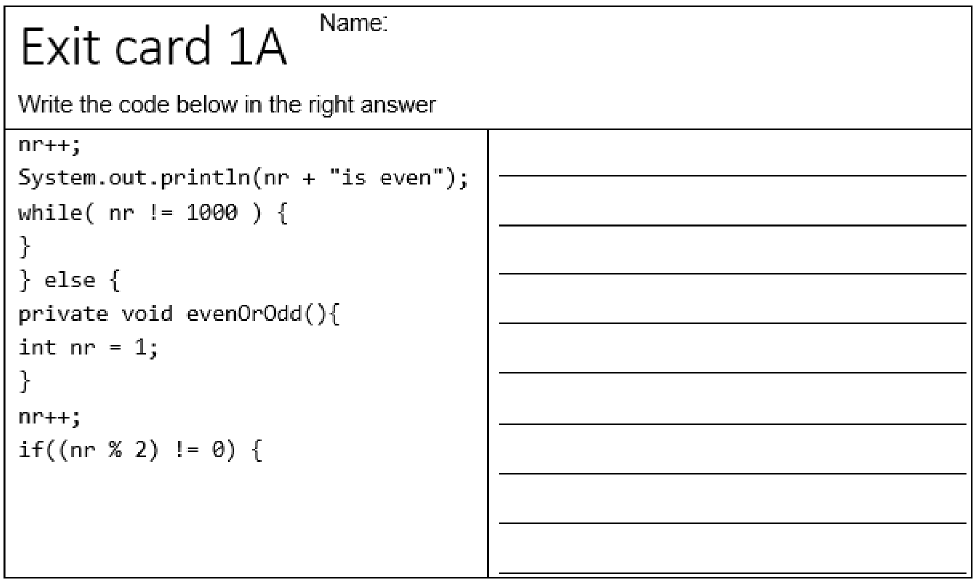
A week later the feedback sessions started. For three consecutive weeks, each student was given an exit card on paper, which they had to work on individually. One of the groups got exit cards which had questions on them in the form of "write code to.." In Figure 3.1 you can find the first exit card with a writing task. The other group got exit cards with Parsons' puzzles


Exit card 1B Name:

Write a method which checks every number from 1 to 1000 if it is even or odd. If the number is even, the print for example "6 is even". If the number is odd, don't print anything. Use a while-loop in your method.

Figure 3.1: The first exit card with a writing task

on them. The writing code exit cards and the Parsons' puzzles exit cards addressed the same tasks. In Figure 3.2 you can find the first exit card with a Parsons' puzzle.


Exit card 1A Name:

Write the code below in the right answer

```
nr++;
System.out.println(nr + "is even");
while( nr != 1000 ) {
}
} else {
private void evenOrOdd(){
int nr = 1;
}
nr++;
if((nr % 2) != 0) {
```

Figure 3.2: The first exitcard with a Parsons' puzzle

During the first session, students complained about writing down all the code. Thus a transition was made from paper to an online assignment. The change for the writing task was from writing with pen on paper, to typing in the answer. The change for the Parsons' puzzle was from writing with pen on paper, to dragging and dropping the lines of code to the right place. We used Microsoft forms for this. All the exit cards are listed in the Appendix at A.3. After the students completed the assignment on the exit card, they were given personal feedback from the teacher. If the student did not do a sufficient job in answering the assignment, they had to do it again using the feedback they had just been given. If the student did give a sufficient answer, they could leave the classroom after they collected their feedback. The audio of the conversations between teacher and student was recorded.

This process was repeated for three consecutive lessons. The lessons were focused on wile-loops, for-loops, lists and building students' understanding of this topic of the curriculum.

After the students got formative feedback for the third time, they were asked to fill in the self-efficacy questionnaire of table 3.1. We used the Motivated Strategies for Learning Questionnaire (MSLQ), developed by García and Pintrich, and adapted it for programming. This questionnaire was designed to measure, among other things, the self-efficacy into a specific subject (García & Pintrich, 1991). We modified the self-efficacy section, which consists of eight questions with a seven-point Likert-scale. The first language of the students is Dutch, therefore the questions were translated from English to Dutch. The Dutch version of the questionnaire can be found in the Appendix at A.8. Scores are calculated by the mean of the items that are included in the scale (eight items for self-efficacy) (Artino & Stephens, 2006).

After each of the three feedback sessions, the teacher of the class wrote down their observations about the session, using the time spent correcting and giving feedback, comments of the students, the answers given on the exit card and the insight in a student's skill. These observations could help in finding things that we may miss in the transcripts of the feedback sessions. They also create an opportunity to gain more insight in how beneficial the Parsons's puzzles can be as a tool to generate feedback.

In the last phase of the study, each group of students was asked to review the assignments and the feedback together using the think-pair-share strategy. We used this method because it enhances their critical thinking ability (Kaddoura, 2013), which increases the quality of their feedback. This reflection/evaluation took place one week after the last formative feedback session. For four questions, students had to write down their own answer,

Name:	Not at all true for me						Very true of me
1. I believe I will receive an excellent grade in the programming part of this course	1	2	3	4	5	6	7
2. I am certain I can understand the most difficult material presented on the readings in programming	1	2	3	4	5	6	7
3. I am confident I can understand the basic concepts taught in programming	1	2	3	4	5	6	7
4. I am confident I can understand the most complex material presented by the teacher in programming.	1	2	3	4	5	6	7
5. I am confident I can do an excellent job on the assignments and tests in programming.	1	2	3	4	5	6	7
6. I expect to do well in the programming part of this course	1	2	3	4	5	6	7
7. I am certain I can master the skills being taught in programming	1	2	3	4	5	6	7
8. Considering the difficulty of programming, the teacher, and my skills, I think I will do well	1	2	3	4	5	6	7

Table 3.1: The self-efficacy questionnaire

compare their answer with another student and then share their thoughts with the entire group. The two groups of students participated in the think-pair-share session separately. So the discussions and their answers were based solely on the type of task they were assigned. These were the questions they were given:

- Did you enjoy the structure of the task?
- Did this task help you understand the given problem better than the other assignment version would?
- Did this task help you understand how code is structured overall?
- Did some misunderstandings you have about code become clear because of this task?

The audio of the entire session was recorded and the written answers were collected.

3.2.3 Data analysis

For the pre-test in the first phase of our research, students' responses were corrected. Based on the percentage of points they earned, they were categorised in one of the following level skills: low(<50%), advanced(<80%) and high(≥80%). We assigned the students to two groups, making sure there are an equal amount of low, advanced and high level students in each group.

In the second phase of our study we gathered information from the feedback sessions with transcripts, a self-efficacy questionnaire and teachers' observations. The transcripts were analysed using $\odot$ and $\otimes$ to mark were the teacher and the student knew if the task was executed correctly or not, respectively. We also used the symbol ▲ to mark feedback that is applicable to only this excercise and the symbol ■ to mark feedback that is applicable to programming in general.

The results of the self-efficacy questionnaire were analysed by calculating the averages.

The teachers' log consists of observations the teacher made during the feedback sessions. Amongst other things, these observations were about how students felt during the tasks, what stood out from the answers the students submitted and how students acted during the tasks and when they got their feedback.

In the third and last phase of this research, the think-pair-share session, we used thematic analysis to identify patterns or themes in our qualitative data generated. Braun and Clarke suggest that it is the first qualitative method that should be learned because it provides core skills that will be useful for conducting many other kinds of analysis (Braun & Clarke, 2006). We also used the guide of Maguire and Delahunt, where they explain how to execute a thematic analysis (Maguire & Delahunt, 2017). Braun and Clarke provided a six-phase guide which we use as a framework for our analysis which you can find in Table 3.2.

Step 1: Become familiar with the data	Step 4: Review themes
Step 2: Generate initial codes	Step 5: Define themes
Step 3: Search for themes	Step 6: Write-up

Table 3.2: Six-phase framework for doing thematic analysis (Braun & Clarke, 2006, p. 3354)

The first step is reading and re-reading the transcripts of the feedback sessions and the think-pair-share session. In the second step we started organising our data in a meaningful and systematic way. We coded the data so it was filtered into small chunks of meaning. We did this by hand and we developed and modified the codes as we worked through the coding process. After performing these two steps we made tables for the codes from think-pair-share session. One table for each of the four questions we asked the students. The codes were sorted in the tables by what kind of task they were about and if they were positive or negative about that task.

The third step focuses on searching for themes in our codes. We use the definition for a theme as a pattern that captures something significant or interesting about the data and/or research question. In the fourth step we reviewed, modified and developed all the themes that we identified before. We looked critically at our themes to make sure they make sense. Then we gathered all the codes that were relevant to each theme Next, we refined our themes in step five. The goal in this step was to identify the essence of each theme. To complete this step we constructed a thematic map. The final step is writing what we learned from our analysis. Which you can find in Chapter 4.

3.3 Ethical Considerations

The students were all 17 years old or older, so we asked for their permission in a consent form. The consent form stated the purpose of the study, what information would be gathered from them and how it would be processed. It also stated that they participated on their free will and could stop with the study if they wanted to at any time. If a participant was not yet 18 years old, their parents were asked to give consent as well. The consent form can be found in Dutch in the appendix A.2.

3.4 Limitations of the study

As mentioned before, our interpretivist research method has some disadvantages which affect the validity of our results. Research outcomes may be affected by the researcher's own interpretation, belief system, ways of thinking or cultural preference which can cause bias (Mack, 2010). However, we were vigilant not to let our preconceptions affect our interpretations. In this study, we demonstrate examples of the transcripts to give the readers the possibility to assess our interpretations.

Furthermore, because we used a qualitative methodology instead of a quantitative methodology we can not rely on numbers of statistics. The results we gathered also cannot be generalized in different environments because of our interpretive research.

Chapter 4

Results and Discussion

In this chapter we present the results we have gathered from our research. We discuss our thoughts about the results and what they could mean from our point of view.

We gathered results from different sources as stated in the methodology chapter corresponding to the three parts of our study. First we present and discuss the results of the pre-test the students did, together with their assignment to one of the two groups. Then we present and discuss some quotes of the students and teacher from the feedback sessions. The third part consists of the discussion and results from the self-efficacy questionnaire we asked the students to fill in. Then we present and discuss our findings from the teachers' log of the feedback sessions. Finally we present and discuss the feedback and opinions we gathered from our students in a think-pair-share session after all the feedback sessions. This also includes a thematic map of our findings from this session.

4.1 Pre-test

The 16 participants of the study performed a test to measure their skill in programming. The test can be found in the Appendix at A.1. In table 4.1 you can find their results.

	Low level (<50%)	Advanced level (<80%)	High level (≥80%)
Group A	3	3	2
Group B	3	3	2
Total	6	6	4

Table 4.1: Number of students of each level in the pre-test

Because of privacy reasons, we only present the levels of the students. The percentage represents how many of the available points on the test a student earned. After the test, we split the students in two groups, to ensure every group had an equal number of low, advanced and high level students. We assigned group A to the Parsons' puzzles and group B to the writing tasks. This split resulted in an average percentage in group A of 64% and an average percentage in group B of 66%. The overall average percentage was 65%.

The skill level differs enormously, although they are evenly spread over the two groups. Some students scored near perfect, while others scored very bad. We believe this can have a big influence on the study. Because group A and group B both contain students with a high skill level and a low skill level, we can gather information about both in our study.

4.2 Feedback sessions

For three consecutive weeks, students were given a programming task on an exit card based on the group they were assigned to after the pre-test. This entails students from group A made exit cards 1A, 2A and 3A and students from group B made exit cards 1B, 2B and 3B. The Dutch transcripts of the first feedback session can be found in the Appendix at A.5. The English translation of these transcripts can be found in the Appendix at A.6. The English translation of the second feedback session can be found in the Appendix at A.7. There are less than 16 conversations each lesson, because some students could not attend them because they were ill or otherwise occupied. We present two conversations from every group.

In these conversations, T stands for Teacher and S stands for student. The conversations are numbered for referencing. The symbol ⊙ marks were the teacher knew if the task was executed correctly or not. The symbol ⊗ marks were the student knew if the task was executed correctly or not. The symbol ▲ marks feedback that is applicable to only this exercise. The symbol ■ marks feedback that is applicable to programming in general.

Feedback session 1 - Student 6 - Parsons' puzzle

T: Nr = 1, while, then you do the if, then print it and then else. Hmm I miss the nr ++. ⊙
S: yes it is here right?
T: yes, but what if. Watch this while here, it checks. Suppose he is odd, then he does the if, then you print it. But this is a check if it is true. ■
S: then it is odd
T: yes

S: ⊗ oh then you also have to do nr ++ there
T: yes exactly and then with that else, what should you do there?
S: then print
T: yes but then you also have to do nr ++ right? ▲ *Otherwise it is odd once and then nothing happens. That's why there are 2 nr ++ things. Okay, thank you.*

Feedback session 2 - Student 7 - Parsons' puzzle

T: Did it work completely? Did you finish the assignment?
S: No, I don't have it yet.
T: ⊙ Oh you haven't done it right.
S: Oh no, I did it completely wrong.
Other student: He always has the prescribed lines, that's not fair at all.
T: There is only one such assignment left, then we will alternate it, okay?
S: ⊗ This line must be back there again.▲
T: Yes.
S: But I actually thought I had done that.
T: Yes I don't know what happened.
S: Well then it is okay.
T: But you get it?
S: Yes, I get it.
T: OK.

Feedback session 1 - Student 5 - Writing task

T: Let's see
S: It's a bit ugly because I didn't know what to do, that's where it starts. And there it goes further
T: okay yes. The smaller than 0, 1000. Then you do the number plus 2 and then print it
S: ⊗ The only mistake is that it doesn't print 0 now, but then it should print here one more time before that. ▲
T: ⊙ Yes exactly
S: But that didn't seem right to me
T: But then you could first print the number and then do the number + 2. ▲
S: Yes I could, but then the good number would have been gone.
T: Further more. If I were you I would take a better look at the assignment next time. It says here that you have to print this. Like 6 = even. ■
S: so number and then string is even.
T: and you have not made a method of it.▲ *But otherwise super, great!*

Feedback session 1 - Student 11 - Writing task

T: yes you have exactly the same thing [as the student before you].
S: Well mine is called nr, that's more practical
T: Let's take a look, private void int numbers. ⊙ *I see you put an ! here first, but now there is only 1 =. And that is an assignment.* ■
S: ⊗ *Ohh yes double =*
T: exactly yes, very good. And it must be comparison. So yes that is number ++. This is unfortunately not the correct syntax for this.
S: okay
T: do you know how ++ works?
S: uhh no. yes that's just the value there
T: yes that's the easy thing about ++. Instead of number = number + 1, you just have to write number ++ and you're done ■
S: okay
T: otherwise completely correct

It was hard to gather useful results from these transcripts. The markers indicating when the teacher and student knew if the task was executed correctly or not did give very interesting results. As you can see in the conversations above, the teacher knew if the answer was correct or not in roughly the same time with Parsons' puzzles as with the writing task. The ▲ and ■ markers indicated more feedback was given on the writing task. The bigger length of the conversations about the writing task indicates this as well. After a writing task, it took longer for the teacher and student to understand everything they did wrong or right then it would for a Parsons' puzzle. The students with a low skill level performing the writing task often needed feedback during the assignment as well. This makes the feedback progress even lengthier. This is not something that is desired in the classroom. Very long personal feedback can cause an unequal distribution of the teacher's attention. In the next two conversations you can see an example of this problem.

Feedback session 2 - Student 3 - Writing task

T: Do you find it harder?
S: Yes
T: Yes you have to write it, right? That is why I have the example on the board. So that you can still look at it.
S: It's really harder.
T: It is exactly the same assignment, but then you have to write it yourself. Try again, you can do it.
[some time passes]

Feedback session 2 - Student 3 & 6 - Writing task

T: If you really can't figure it out, you can indicate that. Then I will help you out.
S: Yes we can't figure it out.
T: Okay then I'm going to help you. Let's see, you are both in the same place in the assignment. You have already created the method, that is very good. We have to print the characters, so you are indeed making a for loop.
S: I only did what is on the board.
T: On the board it is written that we first create a variable. We say the object type and give the variable a name. Then we say that the egg object there on the board is every egg that appears in eggList. And now we have characters. So we actually want to have every character that appears in the list. So we can take over the example almost exactly and adjust it a little bit for our situation. So you said there are strings in that list.
S: Yes.
T: What is the type of the object?
S: String.
T: Exactly. So we can write that down. Open parenthesis, String. And think of a nice name, it doesn't matter. You only need it in the for loop. Nice, good name. Then you put a colon and we need the list that we are going to walk through. That is..
S: Do you have to initialize the characters in the list?
T: Yes well you have to initialize the list first. Because you haven't done that yet.
S: umm ..
T: Shall I give an example of that? How to initialize a list?
S: Yes.
T: Okay [give an example on the board]
S: Should this be in the for loop? Or where should this be?
T: Well you have to initialize it somewhere right? And you said that you already got the list with the method. So you probably have already initialized the list before you enter the method. If you don't get the list, you can simply initialize it in the method. You can then go through the list. Because that can only be done after the initialization. Then, during the process, you can print any letter. Because you always get each letter in the for loop. And that is what you want to print, right?
S: Yes
T: So then you can print that. And then you only have to close the method. Yes?
S: OK.
T: Try this and I'll come back later.

Feedback session 2 - Student 5 - Parsons' puzzle

T: [Name] Are you also going to work on the assignment? Oh you already have it!
S: Yes, look.
T: Oh that's all right. How did it go?
S: It was pretty easy.
T: Yes, nice. Do you think you could have made it before I explained it?
S: Yes.
T: Okay, good. Do you think you could have written it yourself?
S: Yes.
T: OK.

This conversation is about the same programming problem as the conversation before with students 3 and 6. The skill level of students 3, 5 and 6 is also the same. The only difference is student 5 made a Parsons' puzzle and students 3 and 6 made a writing task. In the conversations we can see student 5 found the task "pretty easy" while students 3 and 6 said they "can't figure it out". This indicates students with a low skill level may benefit more from a Parsons' puzzle than a writing task.

Students with a high skill level performing a Parsons' puzzle often did not need any feedback at all. As you can see in the example below.

Feedback session 2 - Student 1 - Parsons' puzzle

T: [Reads aloud] Yes, really good. Did you find the assignment difficult?
S: No.
T: Good.

This indicates students with a high skill level may not benefit as much from a Parsons' puzzle.

The checking of the answers was significantly slowed down if a student wrote down line numbers instead of lines of codes. Students did not want to copy all the lines of code in the right order. So they wrote down line numbers in front of each line and wrote them in the correct order. Because just writing down numbers takes less time and work. The problem is that students did not generate an overview of their code if they use this tactic. Thus making it harder to check their answer. Not only for the teacher, but for themselves as well. This resulted in a lot of wrong answers because of small mistakes which the students probably would not have made if they just copied the line numbers. To make sure this did not happen in the second and third week, we remade the task in digital form. Now the students with the Parsons' puzzles just have to drag and drop the lines in the correct order.

There was one other thing we found during the feedback sessions. Students with writing tasks often delivered an answer which differed from the answer on the answer sheet. And these were often correct as well. This is bound to happen, because every programmer has his or her own style and can come up with a different approach for the same problem. But it did made checking their answers lengthier, because the teacher had to familiarize their code from scratch before they could provide feedback.

This aspect of a writing task, that the solutions can very wildly, is also one of its assets. Students are forced to think of their own approach to the problem. Which makes this a good practice for them. But for students with a low skill level this is sometimes impossible. They do not possess enough knowledge about programming to think of a solution. Which makes this task not very beneficial to their learning. This problem becomes especially clear in the transcripts of the second feedback session.

4.3 Self-efficacy questionnaire

After the students made the third exit card, they filled in a self-efficacy questionnaire which you can find in table 3.1. In figure 4.2 you can find the results of all the participants. One student did not fill in the questionnaire because he or she did not attend the lesson.
For each of the eight statements, the students had to fill in how much they agreed with it on a scale from one to seven. The higher the number, the higher their self-efficacy. In the table you can see the average self-efficacy of every student and for every question. The average self-efficacy score of the students in group A is 4,0. The average self-efficacy score of the students in group B is 5,0. The standard deviation in both groups is 1,0.

At first, we thought the self-efficacy questionnaire would show that students who made Parsons' puzzles have a higher self-efficacy than the students who made writing tasks. Because the Parsons' puzzles were perceived as an easier task and the students making writing tasks struggled with them a lot. But the opposite happened. Students in group A, who made Parsons' puzzles, had the lowest self-efficacy. They differed from the other group with 1 point on a scale of 1 to 7.

The pre-test made sure the group was split in two groups of students that should on average have the same skill level. This was made clear to the students, but during the feedback sessions, students questioned their group assignment a lot. This may have caused the believe in some students that

Participants	Questions								
Group	1	2	3	4	5	6	7	8	Average
A	6	5	7	7	6	6	6	7	6,3
A	4	2	7	4	3	3	4	6	4,1
A	3	2	5	3	2	3	4	3	3,1
A	2	2	3	2	2	3	3	5	2,8
A	3	3	4	4	3	3	3	3	3,3
A	3	4	6	5	3	5	5	6	4,6
A	4	3	2	3	3	3	4	4	3,3
A	4	3	6	3	3	5	6	7	4,6
B	4	2	6	3	4	3	3	7	4,0
B	4	2	3	3	3	4	3	6	3,5
B	5	4	7	5	5	5	6	5	5,3
B	5	4	7	6	6	5	6	6	5,6
B	7	5	7	6	6	7	6	6	6,3
B	6	4	6	5	5	7	5	7	5,6
B	4	5	6	6	3	4	5	6	4,9
A	3,6	3,0	5,0	3,9	3,1	3,9	4,4	5,1	4,0
B	5,0	3,7	6,0	4,9	4,6	5,0	4,9	6,1	5,0
Total	4,3	3,3	5,5	4,3	3,8	4,4	4,6	5,6	4,5

Table 4.2: Results self-efficacy questionnaire

they were assigned to the Parsons' puzzles because they were not skilled enough for the writing task group. The difference could also have been caused by the perceived difference in difficulty of the tasks and how accomplished students could feel after completing a task that is perceived as difficult or easy.

Because the Parsons' puzzles were perceived as easy, students maybe did not think they were good in programming when they could execute them correctly. While the students that made the writing task maybe thought they were good in programming because they could correctly execute the 'hard' task.

4.4 Teachers' log

As a teacher, I made some observations during the feedback sessions. For example, students with Parsons' puzzles were finished earlier than students with a writing task. This indicates that Parsons' puzzles may be quicker to solve than completing a writing task. It is ideal if the task students have to complete does not take a lot of time, because this gives the teacher more opportunities to do it as a formative assessment.

Students were disappointed if they were assigned to group B, because they had to do the 'hard' writing task. They were happy if they were assigned to group A, for the 'easy' Parsons' puzzle. They assigned these difficulties themselves. While I was giving feedback to a student solving a Parsons' puzzle his neighbour voiced the underlined sentence in the following conversation:

> **Feedback session 2 - Student 7 - Parsons' puzzle**
> *T: Did it work completely? Did you finish the assignment?*
> *S: No, I don't have it yet.*
> *T: Oh you haven't done it right.*
> *S: Oh no, I did it completely wrong.*
> *Other student: <u>He always has the prescribed lines, that's not fair at all.</u>*
> *T: There is only one such assignment left, then we will alternate it, okay?*
> *S: This line must be back there again.*
> *T: Yes.*
> *S: But I actually thought I had done that.*
> *T: Yes I don't know what happened.*
> *S: Well then it is okay.*
> *T: But you get it?*
> *S: Yes, I get it.*
> *T: OK.*

As you can see, the student complained because they had to do the writing task, and had to think of all the code by themself. While their neighbour has the Parsons' puzzle, which already provides the lines of code. This indicates that students found Parsons's Puzzles easier and more fun to do. The first is important information for teachers that want to provide students with tasks that match their level. The second part, that Parsons' puzzles could be more fun to do for students is also important. If a teacher has the opportunity to choose between two otherwise equal tasks, they probably want to provide the one that is the most enjoyable for students to do.

Students wrote down line numbers instead of complete sentences of the Parsons' puzzles. Because of this, it was harder to check their answers.

Students also made more mistakes if they did this. An example from the first feedback session, where a student just wrote down the line numbers:

> **Feedback session 1 - Student 4 - Parsons' puzzle**
> *T: Oh you put it like that.*
> *S: Yes, I didn't copy it completely.*
> *T: So we start there, then we do this, then we go here. Do we do that if .. No this is not right*
> *S: Yes, I am sure it is true, ma'am. Why isn't it correct?*
> *T: Yes, let's see, it is indeed not correct*
> *S: Oh wait I put this on the wrong place then. Okay you are right madam, it must be turned around indeed.*
> *T: Yes exactly. Did you just not see this?*
> *S: Yes, I did not take a good look, so it would have been better if I wrote it down, yes.*
> *T: Okay, well. Then you also know what to do next time.*
> *S: Yes*
> *T: Thank you*

As you can see, the student realised they would not have made their mistake if they wrote down the entire answer, because then they would have an overview of their code, which would have made it easier to check. The unwillingness of students to copy lines of code, leads to a higher chance of them making mistakes. But they appear to be willing to take that chance. That's why we found it important to change how the students submit their answers. As told in the methodology at section 3.2.2. We changed the exit cards so students only had to drag and drop the lines of codes, ensuring students did not have the option of only writing down the line numbers.

The answers of the students of the writing task sometimes differed from the answer on the answer sheet, while they were still correct. This made checking the answers harder and take a longer time. Which is not desired, because it uses up valuable time.

Students with all skill levels understood their mistakes from the feedback generated by the Parsons' puzzle very fast. It was easy for the teacher to give feedback based on the Parsons' puzzle. Students with low skill levels required a lot of time to understand the feedback generated by the writing task. It was hard for the teacher to give feedback based on the writing task to them as well. For students with an advanced or high skill level this was not that much of a problem. This indicates that it may be beneficial for students with a low skill level to get feedback on Parsons' puzzles instead of writing tasks.

Students compared their answers during the feedback sessions with the answers of the student that got feedback at that moment.

This means they listened to the feedback given to another student and checked if that feedback was also applicable to their answer. This indicates students have the opportunity to learn from the feedback to other students. Which can save time and creates an extra learning opportunity.

Many different misunderstandings that may indicate students misconceptions were discovered at the Parsons' puzzles. For example, in the first students had to make the following program:

```
private void evenOrOdd(){
    int nr = 1;
    while( nr != 1000 ) {
        if((nr % 2) != 0) {
            nr++;
        } else {
            System.out.println(nr + "= even");
            nr++;
        }
    }
}
```

One student forgot to put `nr++;` after the print statement. Which results in a program that will infinitely loop. Another student put the line `System.out.println(nr + "= even");` before the if clause. Resulting in a program that will print `nr + "= even"` for every `nr`. Sometimes it was harder to discover misunderstandings for the writing task because there were more things which could go wrong.

Students were very happy with the drag and drop version of Parsons' puzzles. This confirms that it was a good decision to change the exit cards from paper to an online task.

All students making the Parsons' puzzles task, performed well. Indicating that they may be too easy for the high level students.

Students making the writing task found it hard. They often could not even make a start with the task. Extra explanation was needed which took a lot of time. The students seemed disengaged and less motivated by the task. Some examples showing this problem:

> **Feedback session 2 - Student 4 - Writing task**
> *S: I really can't figure out what to do.*

T: Have you looked at the example? If you can't figure it out then just show what you did so far. Then I will give feedback about that.
S: OK.

Feedback session 2 - Student 3 & 6 - Writing task
T: If you really can't figure it out, you can indicate that. Then I will help you out.
S: Yes we can't figure it out.
T: Okay then I'm going to help you.
[The teacher explains what the students need to do]

It took a lot of time to explain the problem individually to the students. It seemed the exercise was too hard for these students. There were no students making Parsons' puzzles that needed as much feedback before they could solve it. This indicates a writing task may be too hard for students with a low skill level.

Students making the Parsons' puzzles often could figure out on their own what their mistake was if they delivered the wrong answer. Some examples:

Feedback session 2 - Student 7 - Parsons' puzzle
T: Did it work completely? Did you finish the assignment?
S: No, I don't have it yet.
T: Oh you haven't done it right.
S: Oh no, I did it completely wrong.
Other student: He always has the prescribed lines, that's not fair at all.
T: There is only one such assignment left, then we will alternate it, okay?
S: This line must be back there again.
T: Yes.
S: But I actually thought I had done that.
T: Yes I don't know what happened.
S: Well then it is okay.
T: But you get it?
S: Yes, I get it.
T: OK.

Feedback session 1 - Student 1 - Parsons' puzzle
T: Oh you just wrote down the numbers
S: Yes I have done it with numbers so you have 1 2 3 4
T: Hmm that's a bit hard to check. Where is 1? Oh here and then 1 2 3 4 5 6...
S: That sounded very doubtful

T: Yes, I expected another line first
S: <u>Oh I must have turned them around.</u>
T: Yes. It might be more convenient for next time if you just write it down
S: Yes okay
T: Because I know that it takes less time, but that makes it difficult for me to check. Thus making it more difficult for you to check.
S: Yes that is fine
T: Yes? Okay. Then it's okay.

The part where students figured out their own mistakes are underlined. This indicates students were less dependent on the feedback of a teacher. Which is a desired characteristic because it enhances self-regulated learning, which plays an important role in students' academic achievements (Zimmerman, 1990).

4.5 Think-pair-share session

Seven students from each of the two groups participated in a think-pair-share session. Two students did not participate because they were not present during the lesson in which the sessions was done.

As described in the methodology: we gathered codes from the transcripts and the notes of students of the think-pair-share session. For each of the four questions, we made a table with all the gathered codes for this question.

The codes were put in the column of the kind of task they were about. The rows of the tables indicate whether the codes were negative or positive about the task. If codes appeared more than once, we put the number of times they appeared. In table 4.3 you can also find a row with a suggestion, because this statement was neither positive or negative. For the first question, many codes were gathered because the students provided a lot of feedback. But as the session progressed, the amount of feedback students provided decreased. Therefore, the fourth table contains fewer codes than the first table. This may have happened because students already felt like their feedback about the tasks was already exhausted by the previous questions and they would otherwise repeat themselves. Another reason could be that students felt like they could not answer the question.

4.5.1 Did you enjoy the structure of the task?

	Parsons' puzzle	Writing task
Yes	Feeling more competent Easier to do (3) Diversity in programming tasks Fun to do	
No	Not that challenging (3) Not that useful Too easy to do	Too hard (4) No idea where to begin (3) Unclear goal of the assignment (2) Not fun to do
Suggestions		It should be possible to write answers in an IDE

Table 4.3: Did you enjoy the structure of the task?

In table 4.3, you can see there were no positive remarks about the Writing task. That's why we believe students did not enjoy the writing tasks. They did have a lot of positive remarks for Parsons' puzzles, which indicates that the students did enjoy them. For both of the tasks there were negative remarks, which indicates the students were critical of both.

Almost all of the feedback in this table relates to the difficulty to the task: easy, too easy, too hard. Apparently the difficulty of a task is heavily related to the enjoyment of it. Students with a high skill level found the Parsons' puzzles too easy to do, while Students with a low or advanced skill level found them easier to do in a positive way. The students with a low or advanced skill level who performed writing tasks found them too hard to do. Indicating that matching a students' skill level to the difficulty of a task plays an important role in the enjoyment of a task. Previous work found that two-dimensional puzzles were more complicated than the one-dimensional puzzles we used in our research (Ihantola & Karavirta, 2011). When a programming language based on indentation, like Python, is used, these two-dimensional puzzles could help make the puzzles more challenging for students with a high skill level.

4.5.2 Did this task help you understand the given problem better than the other assignment version would?

	Parsons' puzzle	Writing task
Yes	It made it easy enough to understand the problem	I already understood the problem a little
No	It was confusing I never looked at the problem	Explanation would be better (2) It differs a lot During making the assignment I needed extra help (2)

Table 4.4: Did this task help you understand the given problem better than the other assignment version would?

An interesting code in table 4.4 about Parsons' puzzles is "I never looked at the problem". The student explained it as follows:

> "I just had to look at the set lines of code in the assignment and make sure those lines were in an order that would make a correct program. I did not need to know what problem the code should solve in order to do that."

This indicates that a Parsons' puzzle does not test if a student can solve a programming problem, but if a student can write correct code. While a writing task tests both. Indicating students with a high skill level does not benefit as much by making a Parsons' puzzle.

The other codes are, again, related to the difficulty of the task. Because a Parsons' puzzle already gives you set programming lines, a student with a low skill level commented it gave them the opportunity to understand the problem. While other students with a low skill level who performed writing tasks, stated they could not make the assignment without extra help.

The codes in this table indicate students Parsons's puzzles helped students with a low skill level to understand the given problem better than writing tasks would. For students with a high skill level Parsons' puzzles could fail in helping students understand the given problem.

4.5.3 Did this task help you understand how code is structured overall?

	Parsons' puzzle	Writing task
Yes	You have to think about it (2) Good practice	Good practice (2)
No	It was confusing	I did not have enough understanding (2) The details were hard Not efficient

Table 4.5: Did this task help you understand how code is structured overall?

When we look at table 4.5 we can see that students find both Parsons' puzzles and writing tasks are a good practice for learning how code is structured overall. There is one code that states Parsons' puzzles can confuse your understanding of how code is structured. At first, the lines of the program are in the wrong order, so the student could be confused by this. But this confusion is necessary for the assignment. It is the thing a student should fix to finish the assignment. Other codes stated that Parsons' puzzles are beneficial because you have to think about the code structure so this does not seem to be a significant problem.

And again there are codes about the difficulty of the task. This time the only problem is the difficulty of the writing task. Students with a low skill level said because the task was too hard to execute, it was not beneficial for their understanding in the structure of code. Students with an advanced skill level indicated they found the task not efficient for understanding how code is structured overall. They also said the details of writing code for a writing task is hard. These two codes indicate that even for students with an advanced skill level the difficulty of writing tasks is too high. This may cause the failure of the writing task to help those students understand how code is structured overall.

4.5.4 Did some misunderstandings you have about code become clear because of this task?

	Parsons' puzzle	Writing task
Yes		The feedback was helpful
No	I could not compare the code with how I would do it myself	I needed more help

Table 4.6: Did some misunderstandings you have about code become clear because of this task?

As mentioned before, there are not many codes in table 4.6. Probably because students felt they were repeating themselves or they felt like they could not answer the question. Fortunately, the feedback that was gathered did provide some interesting codes. One code stated an interesting problem with Parsons' puzzles. The set lines of code in the puzzle can differ from the way a student would have written the code themselves. This means that some misunderstandings they might have would never arise because they cannot do them. This can make it hard for a student to translate the feedback they get from this task to how they would write their own code.

This disadvantage is also an asset of Parsons' puzzles. Because it also gives an optimal solution to the student. The lines themselves already are correct so this can help or guide a student in what an optimal solution should look like.

The help a student with a Parsons' puzzle gets from it's task, is the thing a student with a writing task often needs to learn more about there misunderstandings. And with a writing task feedback from the teacher can be crucial. Because the student has to start writing with only a problem description.

4.5.5 Thematic map

Finally, we made a thematic map based on the codes gathered in the think-pair-share session which you can find in figure 4.1.

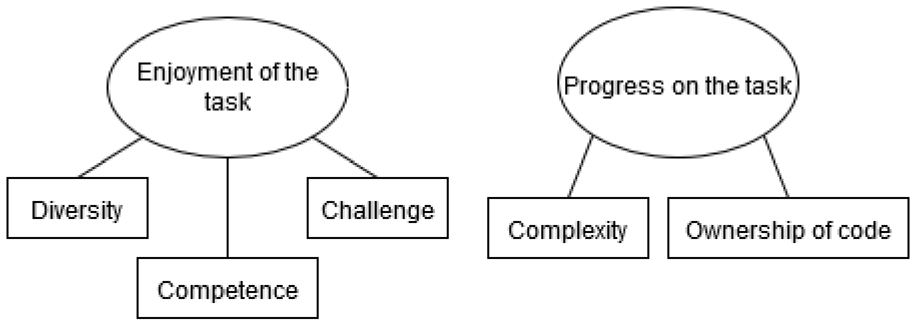


Figure 4.1: Thematic map of feedback from the students

The round boxes are the themes and the square boxes are the sub-themes we gathered from our codes.

The thematic map consists of the two themes we believe are the most important for students: Enjoyment of the task and progress on the tasks. For both of these themes we will discuss how the sub-themes relate to the themes and the codes gathered from the think-pair-share session.

Enjoyment of the task

Students were happy with the diversity of a new kind of task Parsons' puzzles brings. The novelty of it made them more driven to finish the task.

Parsons' puzzles made students with a low skill level feel more competent, because they felt the difficulty of it better suited their skill level, which contributed to their enjoyment of the task.

Students with a high skill level often found the writing task more challenging, which made it enjoyable. While they were not challenged by the Parsons' puzzles, which made those less enjoyable for these students.

Progress on the task

The low complexity of the Parsons' puzzles helped students with a low skill level in understanding the goal of the program. On writing tasks these students needed feedback from the teacher to be able to complete the task, which made it hard for them to use this task to enhance their understanding in code structure. Thus the difference between a writing task and a Parsons' puzzle can have a big impact on the progress of students on their task.

The Parsons' puzzle's set lines of code made it incomparable to their own code, which made students with a high skill level feel like they did not have ownership of the code. It forced them to use the set lines of code, which made these students unable to use their own style of programming, which had a negative impact on their progress. Because of this, these students also felt that a Parsons' puzzle did not help them as much in understanding how code is structured overall and uncovering some misunderstandings they might have, as a writing task could have. While students with a low skill level still lack the need of ownership of their code, which made the set lines of code welcome help.

We believe this thematic map gives a good overview of the two different kinds of assignments and how they could benefit students.

Chapter 5

Conclusions

In this thesis we researched Parsons' puzzles as formative assessment. We did a qualitative study on a class of students in secondary education which we split in two groups. A pre-test ensured we could split the class in such a way, that each group contained students of students with a low, advanced and high skill level in programming. For three consecutive weeks one group made Parsons' puzzles and the other group made writing tasks as formative assessment. We gathered the transcripts from these formative feedback sessions, a self-efficacy questionnaire, a teachers' log. After the feedback sessions were complete, we asked the students what their thoughts were about both kind of tasks in a think-pair-share session.

In this chapter we draw our conclusions about the research questions we stated in the introduction. After this, I reflect on what I've learned from this study as a teacher and how this is going to change my teaching experience. Finally, we propose some potential future work to end our conclusion.

5.1 Conclusion

Are Parsons' puzzles beneficial for teachers as a formative assessment?

During our thesis we could only gather results from the perspective of one teacher. Thus this answer is limited to only my reflections and observations as a teacher. The results of my thesis conclude there is not a concrete answer to this question. Future research should explore this phenomenon.

My first observation is that it takes less time to correct a Parsons' puzzle than to correct a writing task, because the amount of possible answers is smaller. This is supported by previous research (Ericson et al., 2017). Because it is desirable for teachers to spend as little time as possible on correcting answers, this makes Parsons' puzzles a better alternative.

Next is the unequal distribution of the teachers' attention on students making writing tasks because of the lengthy feedback students with a low skill level need. However, we should acknowledge that we observed that students with a high skill level found the tasks too easy. Which could potentially make the lessons less challenging for them.

Are Parsons' puzzles beneficial for students to enhance their understanding in programming?
For this sub-question we can argue that for students with a low skill level in programming, the lower difficulty of Parson's gives them a better opportunity to enhance their understanding. When these students made writing tasks, the goal of the task was often unclear enough for them to enhance their understanding. Previous research found Parsons' puzzles reduce the cognitive load and that exercises with a high cognitive load can slow gaining expertise (Ericson et al., 2017)(Sweller, 1988). This supports our findings because the exercise has a higher cognitive load for students with a low skill level as opposed to students with a high skill level.

For the students with an advanced skill level, unfortunately no results were evident from our study. But, students with a high skill level were denied the opportunity to think of their own approach to the given problem when they made a Parsons' puzzle, because of the given set of lines of code. While writing tasks do not have this restriction.

The answers to these sub-questions help us answer our research question:
Can Parsons' puzzles be used as a tool to generate effective formative assessment?
We believe that when a teacher keeps in mind the difference in difficulty of Parsons' puzzles and writing tasks, and matches these accordingly to the skill level of their students, Parsons' puzzles can be a useful tool to generate effective formative assessment.

5.2 Reflection

In the introduction I reflected on my experiences as a teacher of computer science in secondary education. I stated three problems I encountered while assessing their skill level using writing tasks: The lack of an IDE when writing on paper, the high cognitive load of the tasks and the difficulty of giving good feedback.

During this study I learned new things, because it forced me to reflect on my teaching. But the most important lessons I learned were from the feedback my students gave me. They were very honest in their opinion and I found it good to see how good they were at giving constructive feedback about their learning experience. I will definitely keep asking for feedback from my students, so I can keep on learning from them.

Another thing I learned from this study, is that Parsons' puzzles are a good alternative for writing tasks for students with a low skill level in programming. The problem of the high cognitive load of the writing tasks and the difficulty in giving good feedback about them to these students can be solved by using Parsons' puzzles.

Because the goal of formative assessment is giving students useful feedback on their learning, not a grade, it does not matter that students perform tasks with a different difficulty. In the future, I will adapt my formative assessment methods to a students' skill level. With two kinds of tasks for students with a low skill level and a high skill level, so students can choose between the two tasks. This will give students with a low skill level more opportunity to get useful feedback, which, I hope, will result in them gaining programming skill faster. This also ensures students with a high skill level will not suffer from lack of time I have to engage with them, because of the extra explaining I have to give the slower students. In conclusion, I believe giving students assignments as formative assessment that match their difficulty will improve the quality of the formative assessment in my classroom.

5.3 Future work

Because this is a qualitative research, we can not rely on numbers of statistics. Future work could extent this study by employing more students, of different ages and different stages of learning how to program. Also, other studies could explore other teachers' perspectives and have more insights into how we can improve the quality of the feedback on both Parsons' puzzles and on writing tasks. Our last recommendation for future work is to investigate how Parsons' puzzles could affect students' self-efficacy, because the results from our questionnaire was the opposite of what we expected.

References

Artino, A., & Stephens, J. M. (2006). Learning online: Motivated to self-regulate. *Academic Exchange Quarterly*, *10*(4), 176–182.

Atkinson, R. K., Catrambone, R., & Merrill, M. M. (2003). Aiding transfer in statistics: Examining the use of conceptually oriented equations and elaborations during subgoal learning. *Journal of Educational Psychology*, *95*(4), 762.

Bandura, A. (2006). Guide for constructing self-efficacy scales. *Self-efficacy beliefs of adolescents*, *5*(1), 307–337.

Black, P., & Wiliam, D. (1998). *Inside the black box: Raising standards through classroom assessment.* Granada Learning.

Black, P., & Wiliam, D. (2009, Jan 23). Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability(formerly: Journal of Personnel Evaluation in Education)*, *21*(1), 5. Retrieved from `https://doi.org/10.1007/s11092-008-9068-5`

Bloom, B. S., Krathwohl, D. R., & Masia, B. B. (1984). Bloom taxonomy of educational objectives. In *Allyn and bacon.* Pearson Education.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, *3*(2), 77–101.

Cauley, K. M., & McMillan, J. H. (2010). Formative assessment techniques to support student motivation and achievement. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas*, *83*(1), 1-6. Retrieved from `https://doi.org/10.1080/00098650903267784`

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research* (pp. 113–124). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1404520.1404532`

Ericson, B. J., Margulieux, L., & Rick, J. (2017). Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th koli calling international conference on computing education research* (pp. 20–29). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/3141880.3141895`

Fattig, M. L., & Taylor, M. T. (2007). *Co-teaching in the differentiated classroom: successful collaboration, lesson design, and classroom man-*

agement, grades 5-12. John Wiley & Sons.

García, T., & Pintrich, P. R. (1991). Student motivation and self-regulated learning: A lisrel model.

Hammersley, M. (2012). *What is qualitative research?* A&C Black.

Harms, K. J., Chen, J., & Kelleher, C. L. (2016). Distractors in parsons problems decrease learning efficiency for young novice programmers. In *Proceedings of the 2016 acm conference on international computing education research* (pp. 241–250). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2960310.2960314`

Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, *77*(1), 81-112. Retrieved from `https://doi.org/10.3102/003465430298487`

Helminen, J., Ihantola, P., Karavirta, V., & Alaoutinen, S. (2013, March). How do students solve parsons programming problems? – execution-based vs. line-based feedback. In *2013 learning and teaching in computing and engineering* (p. 55-61).

Ihantola, P., & Karavirta, V. (2011). Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education*, *10*, 119–132.

Kaddoura, M. (2013). Think pair share: A teaching learning strategy to enhance students' critical thinking. *Educational Research Quarterly*, *36*(4), 3–24.

Kluger, A., & DeNisi, A. (1996, 03). The effects of feedback interventions on performance: A historical review, a meta-analysis, and a preliminary feedback intervention theory. *Psychological Bulletin*, *119*, 254-284.

Mack, L. (2010). *The philosophical underpinnings of educational research.* Polyglossia.

Maehr, M. L., & Anderman, E. M. (1993). Reinventing schools for early adolescents: Emphasizing task goals. *The Elementary School Journal*, *93*(5), 593–610.

Maguire, M., & Delahunt, B. (2017). Doing a thematic analysis: A practical, step-by-step guide for learning and teaching scholars. *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education*, *9*(3).

Morrison, B. B., Margulieux, L. E., Ericson, B., & Guzdial, M. (2016). Subgoals help students solve parsons problems. In *Proceedings of the 47th acm technical symposium on computing science education* (pp. 42–47). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2839509.2844617`

Nicol, D. J., & Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: a model and seven principles of good feedback practice. *Studies in Higher Education*, *31*(2), 199-218. Retrieved from `https://doi.org/10.1080/03075070600572090`

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. In *Proceedings of the 8th australasian conference on computing education - volume 52* (pp. 157–163). Darlinghurst, Australia, Australia: Australian Computer Society, Inc. Retrieved from `http://dl.acm.org/citation.cfm?id=1151869.1151890`

Patka, M., Wallin-Ruschman, J., Wallace, T., & Robbins, C. (2016). Exit cards: creating a dialogue for continuous evaluation. *Teaching in Higher Education*, *21*(6), 659-668. Retrieved from `https://doi.org/10.1080/13562517.2016.1167033`

Pham, L. (2018). A review of key paradigms: positivism, interpretivism and critical inquiry. *University of Adelaide*.

Robertson, K. (2006). Increase student interaction with "think-pair-shares" and "circle chats". Retrieved from `https://www.colorincolorado.org/article/increase-student-interaction-think-pair-shares-and-circle-chats`

Stiggins, R. (2005). From formative assessment to assessment for learning: A path to success in standards-based schools. *Phi Delta Kappan*, *87*(4), 324-328. Retrieved from `https://doi.org/10.1177/003172170508700414`

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, *12*(2), 257-285. Retrieved from `https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1202_4`

Tuli, F. (2010). The basis of distinction between qualitative and quantitative research in social science: Reflection on ontological, epistemological and methodological perspectives. *Ethiopian Journal of Education and Sciences*, *6*(1).

van Dinther, M., Dochy, F., & Segers, M. (2011). Factors affecting students' self-efficacy in higher education. *Educational Research Review*, *6*(2), 95 - 108. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1747938X1000045X`

Zimmerman, B. J. (1990). Self-regulated learning and academic achievement: An overview. *Educational psychologist*, *25*(1), 3–17.

Appendix A

Appendix

A.1 Pretest (Dutch)

Toets Informatica V6

Datum: oktober 2019 Duur: 60 minuten

Stof: Hoofdstukken 1 t/m 5 van Algoritmisch Denken op Informatica-Actief

Er zijn 10 opdrachten waarvoor je in totaal 29 punten halen. Bij elke opdracht staat aangegeven hoeveel punten je daarvoor kunt halen. Als de punten niet gelijk verdeelt zijn over de deelopdrachten staat het daar ook bij.

Opdracht 1 – (2pt.)

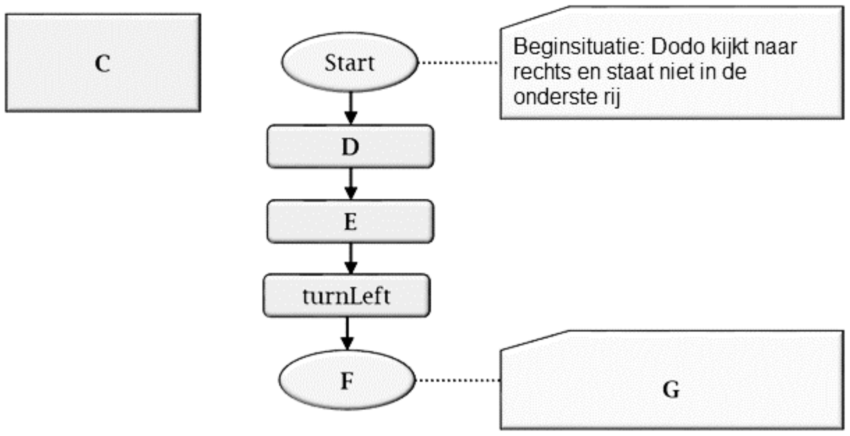
Welke van de volgende zijn een int?

A. getNrOfEggsHatched()
B. 4
C. 3.2
D. vier
E. -3
F. 0
G. False

Opdracht 2 (3pt.)

Hieronder staat code voor een nieuwe methode en het bijbehorende stroomdiagram.
Geef voor A t/m G aan wat er in de code of het stroomdiagram moet staan.
Bij de code moet je de juiste syntax gebruiken.

```
public A moveDown ( ) {
     turnRight( );
     move( );
     B
}
```


C
Start
Beginsituatie: Dodo kijkt naar rechts en staat niet in de onderste rij
D
E
turnLeft
F
G

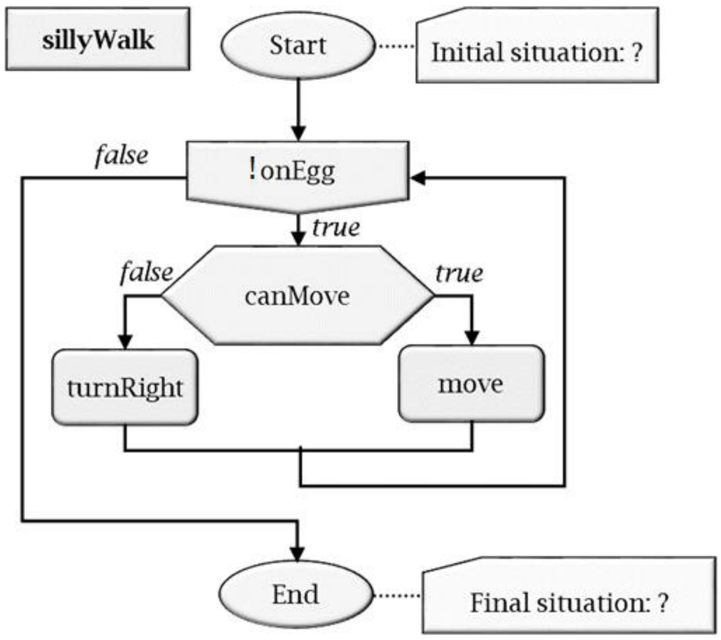
Opdracht 3 – (2pt.)

Stel MyDodo heeft de volgende methode: `boolean alleEierenGevonden( )`, die aangeeft of de dodo al haar eieren gevonden heeft. Hieronder staan een aantal stellingen. Geef per stelling aan of deze waar of onwaar is.

a) Deze methode heeft een boolean parameter.
b) De begin- en eindsituatie van de methode zijn aan elkaar gelijk.
c) Deze methode levert een boolean op.
d) Dit is een mutator methode.

Opdracht 4 – (5pt.)

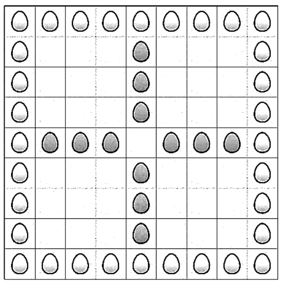
Bekijk het stroomdiagram hieronder. Schrijf de programmacode en voeg een beschrijving van de beginsituatie en eindsituatie toe aan jouw (JavaDoc) commentaar.



Opdracht 5 – (6pt.)

We willen dat Dodo het eierenpatroon hiernaast maakt. Daarvoor mogen we de volgende methodes gebruiken:

Methode name	Methode Omschrijving
`jump ( int n )`	Ga *n* cellen vooruit
`goldenEggTrail(int n)` ○	Legt een spoor van *n* gouden eieren. Het eerste ei ligt in de huidige cel. Na afloop staat Dodo direct achter het laatste ei.
`blueEggTrail(int n)` ●	Zelfde als goldenEggTrail maar dan met blauwe eieren.
`turnRight ( )`	Draai 90 graden met de klok mee.

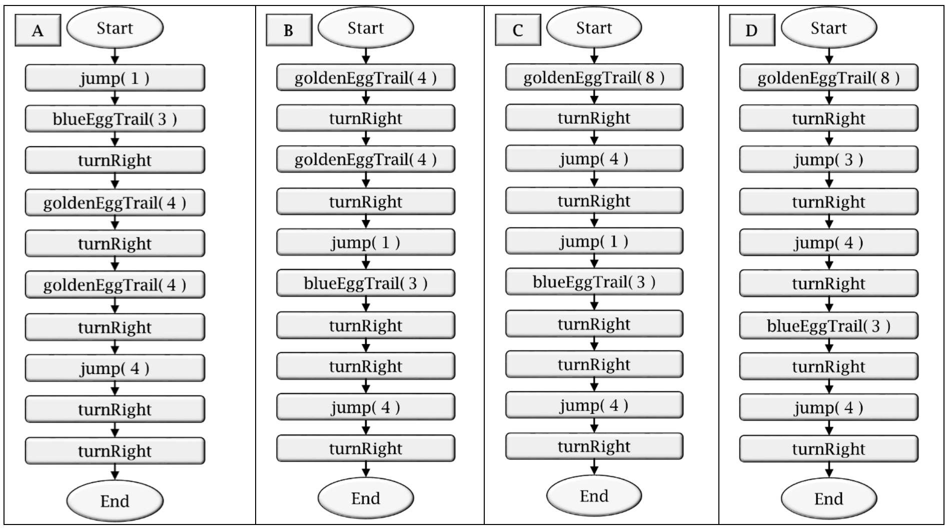


Voorbeeld: Stel Dodo staat in de linkerbovenhoek. Dan kunnen we de rand van gouden eieren uit het plaatje maken door de volgende code 4 keer achter elkaar uit te voeren:

```
goldenEggTrail( 8 );
turnRight();
```

We hebben voor het gewenste patroon de volgende vier oplossingen bedacht. Elk van deze oplossingen wordt **vier keer achter elkaar uitgevoerd**. Je mag aannemen dat de startpositie van Dodo zodanig gekozen is dat het patroon precies in de wereld past.

a) [2pt.] Een van deze oplossingen is fout. Welke is dat?



b) [2pt.] Corrigeer de foutieve oplossing door op de juiste plek in het stroomdiagram precies één instructie toe te voegen. Geef hiervoor aan tussen welke instructies je een wijziging wilt doen en wat de nieuwe instructie is.
c) [2pt.] Of het algoritme werkt hangt ook af van de startpositie van Dodo. Ga ervan uit dat Dodo aan het begin naar het oosten kijkt. Geef voor alle vier de algoritmes, de coördinaten van de cel waar Dodo moet beginnen.

Opdracht 6 – (2pt.)

```
int punten = 4;
if (punten == 3 && punten > 1){
      System.out.println( "A" );
}else{
      System.out.println( "B" );
}
if (punten > 2 && punten < 2){
      System.out.println( "C" );
}elif (punten == 4 || punten <=3){
      System.out.println( "D" );
}
if (punten != 1){
      System.out.println( "E" );
}else{
      System.out.println(punten);
}
```

a) Wat print het programma?
b) Stel we veranderen de eerste regel in `int punten = 1;` Wat print het programma dan?

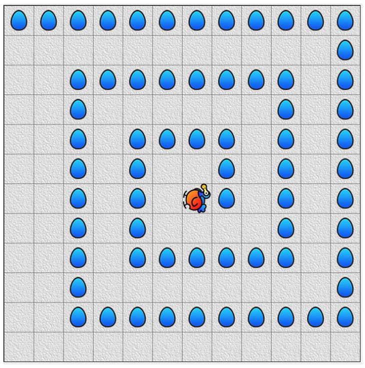
Opdracht 7 – (2pt.)

```
1.  private int surprise (int n, int m) {
2.      int y = 0;
3.      int x = 0;
4.      while (x < n) {
5.          y = y + m;
6.          x = x + 1; }
7.      return y; }
```

Neem onderstaande tracing table over en vul hem aan voor `surprise(3, 5)`.

Regel	n	m	x	y
1			Geen waarde	Geen waarde
2			Geen waarde	
..				

Opdracht 8 – (4pt.)



Maak een algoritme waarmee het spiraal-patroon hier beneden gemaakt wordt. Je algoritme moet voor elke wereldgrootte werken.
Je mag van het volgende uitgaan:

- Er is een variabele `WORLD_WIDTH` eerder in de code gedeclareerd die aangeeft hoe breed de wereld is.
- De wereld is even hoog als breed.
- Dodo begint in de linkerbovenhoek, kijkend naar rechts.
- Dodo eindigt op een ei.

Geef de methode, en let op de correcte syntax.
Je hoeft geen beschrijving van de begin- en eindsituatie te geven.

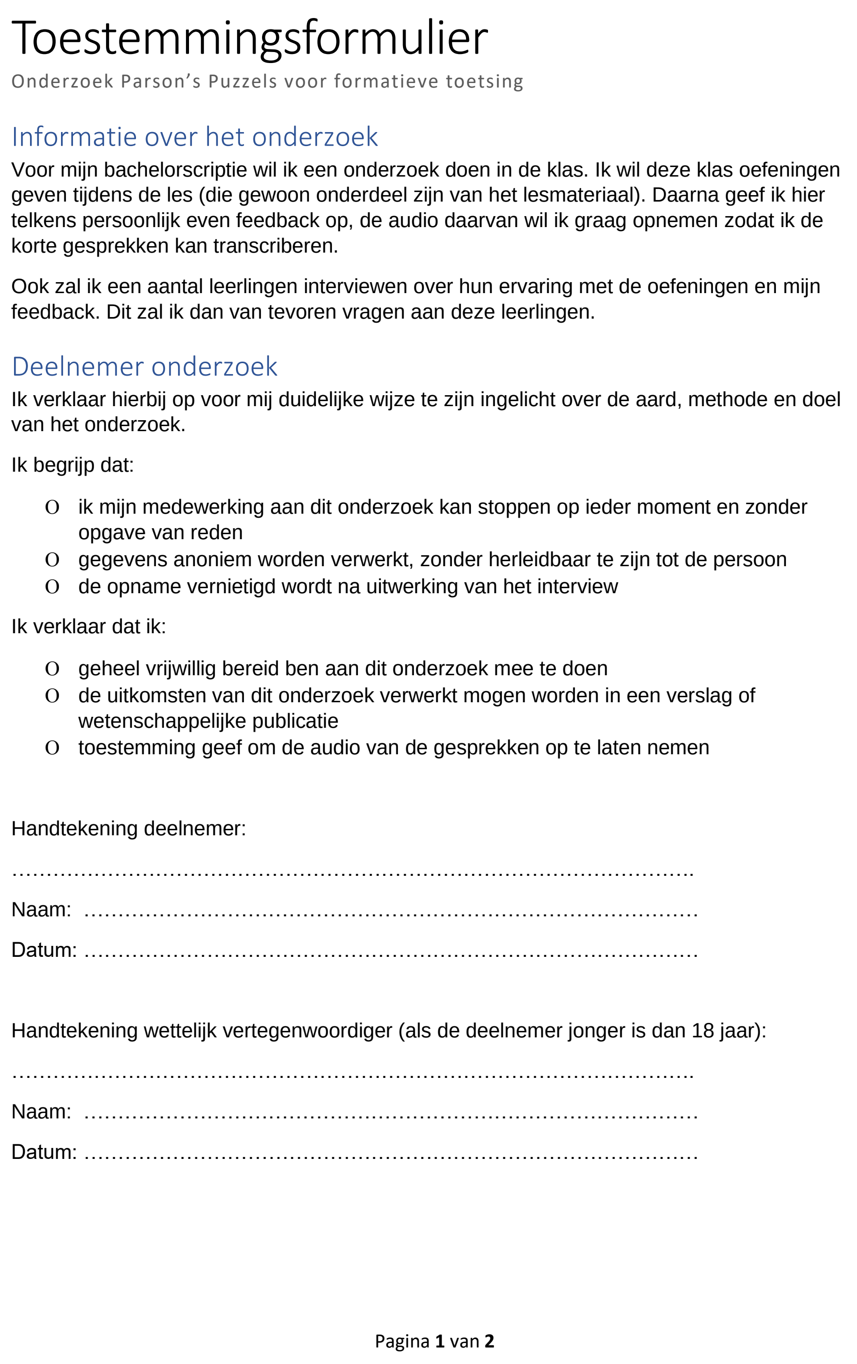
Tip: gebruik `blueEggsTrail(int n)` van opgave 6, `turnRight()` en `layEgg()`.

Opdracht 9 – (3pt.)

```
1.   public double AverageofList(List<Integer> list ){
2.      //Declare the internal variables
3.      List<Integer> listOfNumbers = list;
4.      int sum = 0;
5.      int count = 0.0;
6.      double avg = 0.0;
7.
8.      //Count how many positive numbers there are in the list and
9.      //calculate the sum of all the positive numbers in the list.
10.     for (int number : listOfNumbers){
11.         if( number > 0){
12.             sum++;
13.             count++;
14.         }
15.     }
16.     //only try to calculate the average if there is at least
17.     //one positive number
18.     if ( count >= 0){
19.         avg = (double) sum/count;
20.     }
21.     return avg;
22.  }
```

Leerlingen hebben de opdracht gekregen een methode te schrijven die het gemiddelde van alleen de positieve getallen in een lijst berekent. Hierbij is 0 geen positief getal. Er staan echter drie fouten in. Geef voor elke fout het regelnummer en een verbetering.

A.2 Consent form (Dutch)



Toestemmingsformulier

Onderzoek Parson's Puzzels voor formatieve toetsing

Informatie over het onderzoek

Voor mijn bachelorscriptie wil ik een onderzoek doen in de klas. Ik wil deze klas oefeningen geven tijdens de les (die gewoon onderdeel zijn van het lesmateriaal). Daarna geef ik hier telkens persoonlijk even feedback op, de audio daarvan wil ik graag opnemen zodat ik de korte gesprekken kan transcriberen.

Ook zal ik een aantal leerlingen interviewen over hun ervaring met de oefeningen en mijn feedback. Dit zal ik dan van tevoren vragen aan deze leerlingen.

Deelnemer onderzoek

Ik verklaar hierbij op voor mij duidelijke wijze te zijn ingelicht over de aard, methode en doel van het onderzoek.

Ik begrijp dat:

O ik mijn medewerking aan dit onderzoek kan stoppen op ieder moment en zonder opgave van reden
O gegevens anoniem worden verwerkt, zonder herleidbaar te zijn tot de persoon
O de opname vernietigd wordt na uitwerking van het interview

Ik verklaar dat ik:

O geheel vrijwillig bereid ben aan dit onderzoek mee te doen
O de uitkomsten van dit onderzoek verwerkt mogen worden in een verslag of wetenschappelijke publicatie
O toestemming geef om de audio van de gesprekken op te laten nemen

Handtekening deelnemer:

……………………………………………………………………………………….

Naam: ………………………………………………………………………………

Datum: ………………………………………………………………………………

Handtekening wettelijk vertegenwoordiger (als de deelnemer jonger is dan 18 jaar):

……………………………………………………………………………………….

Naam: ………………………………………………………………………………

Datum: ………………………………………………………………………………

Pagina **1** van **2**

Onderzoeker

Ik heb mondeling toelichting verstrekt over de aard, methode en doel van het onderzoek. Ik verklaar mij bereid nog opkomende vragen over het onderzoek naar vermogen te beantwoorden.

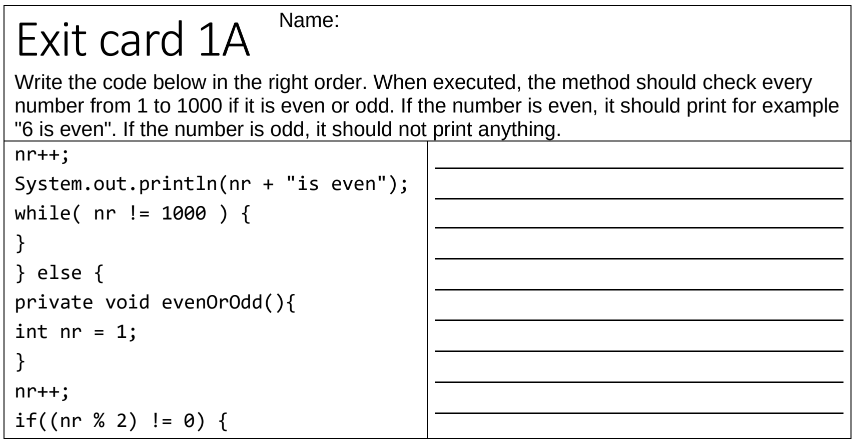
Handtekening: …………………………………………………………………..

Naam: ……………………………………………………………………………

Datum: ……………………………………………………………………………

A.3 Exit cards

Exit cards 1A, 2A and 3A, containing Parsons' puzzles, were handed out in that order to the students in group A. Exit cards 1B, 2B and 3B, containing writing tasks, were handed out in that order to the students in group B.

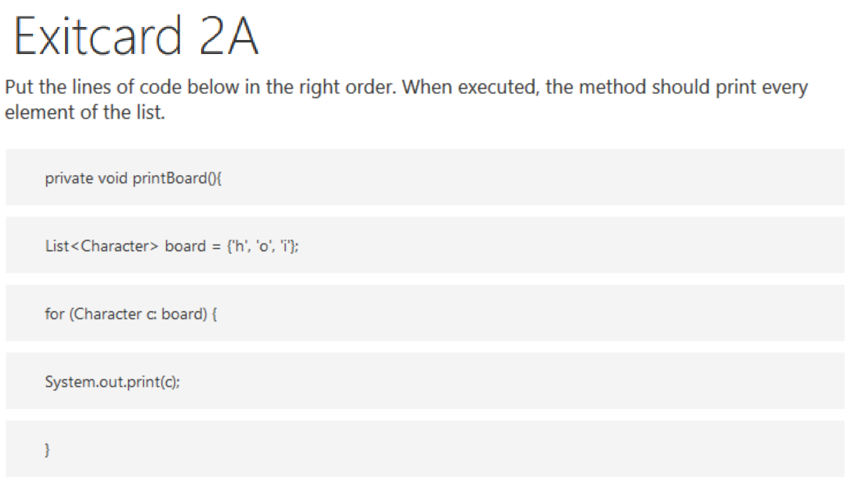


Exit card 1A

Name:

Write the code below in the right order. When executed, the method should check every number from 1 to 1000 if it is even or odd. If the number is even, it should print for example "6 is even". If the number is odd, it should not print anything.

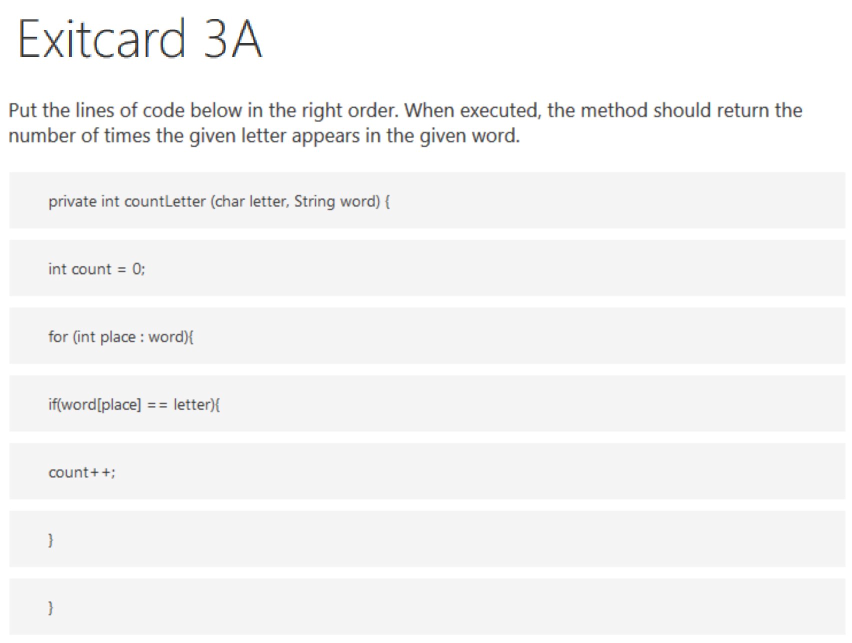
```
nr++;
System.out.println(nr + "is even");
while( nr != 1000 ) {
}
} else {
private void evenOrOdd(){
int nr = 1;
}
nr++;
if((nr % 2) != 0) {
```



Exitcard 2A

Put the lines of code below in the right order. When executed, the method should print every element of the list.

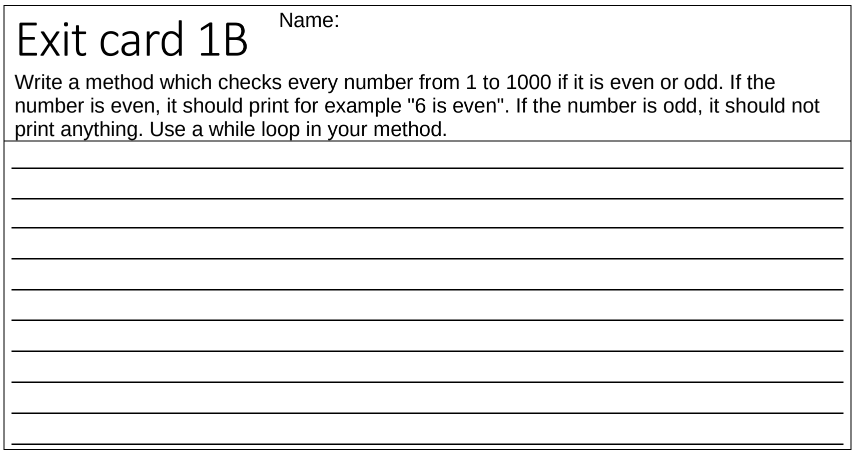
```
private void printBoard(){

List<Character> board = {'h', 'o', 'i'};

for (Character c: board) {

System.out.print(c);

}
```



Exitcard 3A

Put the lines of code below in the right order. When executed, the method should return the number of times the given letter appears in the given word.

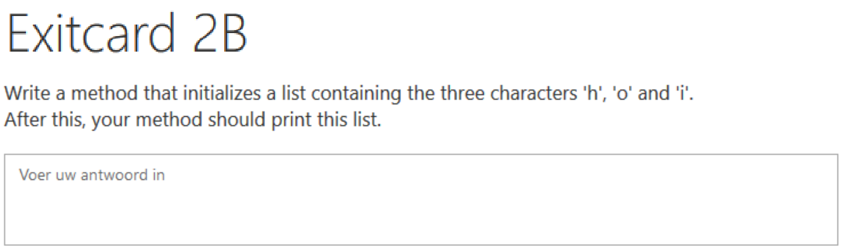
```
private int countLetter (char letter, String word) {

int count = 0;

for (int place : word){

if(word[place] == letter){

count++;

}

}
```



Exit card 1B

Name:

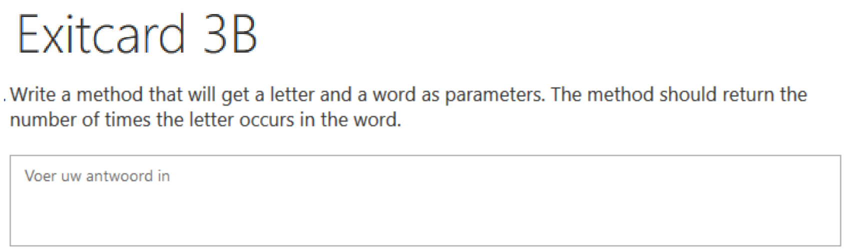
Write a method which checks every number from 1 to 1000 if it is even or odd. If the number is even, it should print for example "6 is even". If the number is odd, it should not print anything. Use a while loop in your method.



Exitcard 2B

Write a method that initializes a list containing the three characters 'h', 'o' and 'i'.
After this, your method should print this list.

Voer uw antwoord in



Exitcard 3B

. Write a method that will get a letter and a word as parameters. The method should return the number of times the letter occurs in the word.

Voer uw antwoord in

A.4 Exit card answers

These are the answers to the exit cards from A.3. The answer for a writing task and Parsons' puzzle with the same number, is the same.

Exitcard 1

```
private void evenOrOdd(){
  int nr = 1;
  while( nr != 1000 ) {
    if((nr % 2) != 0) {
      nr++;
    } else {
      System.out.println(nr + "= even");
      nr++;
    }
}
```

Exitcard 2

```
private void printBoard(){
  List<Character> board = {'h', 'o', 'i'};
  for (Character c: board) {
    System.out.println(c);
  }
}
```

Exitcard 3

```
private int countLetter (char letter, String word) {
  int count = 0;
  for (int place : word){
    if(word[place] == letter){
      count++;
    }
  }
  return count;
}
```

A.5 Transcripts feedback session 1 (Dutch)

T stands for Teacher, S stands for student. Before each conversation, the group the student is assigned to, is stated. The students are numbered for referencing. The symbol ⊙ marks were the teacher knew if the task was executed correctly or not. The symbol ⊗ marks were the student knew if the task was executed correctly or not.

Student 1 - Parsons' puzzle

T: Oh je hebt alleen de nummers opgeschreven
S: Ja ik heb met nummertjes gedaan dus je hebt zo 1 2 3 4
T: Hmm dat is een beetje moeilijk nakijken. Waar is 1? Oh hier en dan 1 2 3 4 5 6... ⊙
S: Dat klonk heel erg twijfelachtig
T: Ja ik verwachte daar eerst nog een else
S: ⊗ Oh dan had ik ze omgedraaid
T: Ja. Het is misschien handiger voor de volgende keer als je het gewoon opschrijft
S: Ja oke
T: Want ik weet dat dit minder tijd kost, maar zo is het voor mij moeilijk te controleren en voor jou ook moeilijk om te controleren.
S: Ja is goed
T: Ja? Oke. Dan is het verder goed

Student 2 - Parsons' puzzle

T: En even zien
S: We beginnen met de private void
T: Oh je hebt..
S: we beginnen met 6 en dan gaan we naar 7 en dat is het laatste
T: 6, 7, 3. Hmm ik heb eerst nog.. Wanneer initialiseer je het nummer? Oh die doe je inderdaad daarna, 6 7 3 is de while. Dan doen we de if, 10 9, ja die is goed. Dan komt de else
S: en dan komt de fout ⊗
T: je bent bij 1 en 2. Huh hmm
S: Jaaa hier komt de fout
T: Ja precies. ⊙ Zie je zelf dat je het fout hebt gedaan?
S: Ik zie het ja
T: Ja je moet dus het printen als die even is. Dan moet je hem dus niet voor elk getal printen. Dat hij die wel doet. Dus dat is een beetje
S: Ja
T: Verder begrijp je wel welke fout je hebt gemaakt?
S: Ja

Student 3 - Parsons' puzzle

T: Oh ja je hebt het ook op dezelfde manier overgeschreven. Misschien voor jezelf ook fijn. Schrijf het programma gewoon op. Je merkt al dat het voor mij moeilijk is om te controleren, en dan is het voor jezelf ook moeilijk om te controloren. Dus je kunt het zelf makkelijker checken en even nalopen of het goed is gegaan. Ja? Oke top
S: dus er staan nog int = 0
T: Jaaa goedzo even zien. Oh die int, ja. Hij perfect! $\odot$ Goed gedaan! Was het nog moeilijk?
S: Nee $\otimes$
T: Oke top, goedzo!

Student 4 - Parsons' puzzle

T: Oh je hebt zo
S: Ja ik heb het niet helemaal overgeschreven.
T: Dus we beginnen daar, dan doen we dit, dan gaan we hier heen. Doen we die if.. $\odot$ Nee het klopt niet
S: Jawel ik weet zeker dat het klopt mevrouw. Hoezo klopt die niet?
T: Ja even kijken, het klopt inderdaad niet
S: Oh wacht ik heb deze bij de verkeerde gezet dan. $\otimes$ Oke u heeft gelijk mevrouw, die moet omgedraaid inderdaad.
T: Ja precies. Had je deze gewoon niet gezien?
S: Ja ik had verkeerd gekeken, dus ik had het inderdaad beter kunnen opschrijven, maarja.
T: Oke, nou goed. Dan weet je ook weer wat je de volgende keer moet doen.
S: Ja
T: Dankjewel

Student 5 - Writing task

T: Even zien
S: Een beetje lelijk want ik wist niet goed wat ik moest doen, daar begint die. En daar gaat die verder
T: oke ja. De kleiner dan 0, 1000. Dan doe je het getal plus 2 en dan print die het getal
S: Het enige foutje is dat die nu niet 0 uitprint maar dan moet die eigenlijk nog een keer daarvoor hier al uitprinten. $\otimes$
T: $\odot$ Ja precies
S: Maar dat leek me niet goed
T: Maar dan zou je ook eerst het getal kunnen printen en daarna pas getal + 2 kunnen doen.
S: Jaa dat zou ook kunnen, dan was het verder wel met het goede aantal gegaan

T: Verder. Ik zou even beter naar de opdracht de volgende keer. Hier staat dat je dit moet printen. Zoals 6 = even.
S: dus getal en dan string is even.
T: en je hebt er geen methode van gemaakt. Maar verder super, prima!

Student 6 - Parsons' puzzle

T: jij hebt hier die gedaan. Nr = 1, while, dan doe je de if, dan print ie het en else. Hmm ik mis de nr++ ⊙
S: ja die staat hier toch?
T: ja, maar wat als. Kijk deze while hier, die checkt. Stel hij is oneven, dan doe hij de if, dan print jij hem. Maar dit is dus een check, als deze true is
S: dan is hij oneven
T: ja
S: ⊗ oh dan moet je daar ook nr++ doen
T: ja precies en dan bij die else, wat moet je daar doen?
S: dan printen
T: ja maar dan moet je ook nr++ doen toch? Anders is het eenmaal oneven en dan gebeurt er niks. Daarom zijn er 2 nr++ dingen. Oke dankjewel

Student 7 - Writing task

T: public int print getallen en dan gaat hij punten returnen. Het getal is kleiner dan 1000. Getal is even. ⊙ Je doet geen check. Je begint bij 2. Dan print je 2 is even en dan is getal 3. En dan print je het weer.
S: ++ is toch gewoon, wacht dat is maar 1 er bij
T: Ja het is niet helemaal de correcte syntax zie je dat? Als je dit doet [schrijft voorbeeld op] dan doet hij += 1
S: En wanneer je
T: voor += 2 kun je twee keer die regel doen, of gewoon += 2
S: ⊗ ooohh +=, oh ja. Moet ik het nu helemaal opnieuw gaan schrijven?
T: nee als je het begrijpt is het goed.
S: Jaaa oke. Dan prima!

Student 8 - Parsons' puzzle

T: void evenOrOdd, int nr 1. 1000
S: ja dan moet er 2x nr++ staan, hier ook.
T: Ja precies. Oh dan heb je het echt heel goed gemaakt! ⊙ Je bent alleen die nr++ vergeten.
S: ⊗ ja die had ik maar 1 x gezien, dus die heb ik ook niet 2x opgeschreven
T: Ja, oke. Goedzo!

Student 9 - Parsons' puzzle

T: private void evenOrOdd, nr = 1. Ja. Dan print hij, oei. ⊙ Heb je goed gekeken naar deze vergelijking?
S: Waarschijnlijk niet.
T: Hier staat, nr %2 . Dus als hij modulo 2 ongelijk is aan 0, dan komt er waarschijnlijk dus 1 uit
S: En dan is het oneven, aha. ⊗
T: Dus iets kritischer kijken naar de code. Maar verder is hij prima!

Student 10 - Writing task

T: methode erop is ook goed. Eerst even de variabele met value initialiseren. Als hij kleiner of gelijk aan 1000 is dan gaat hij door. Als hij ongelijk is aan 1, dan is hij 0 dus dan is hij even. Ja dat klopt. En dat is het enige wat je checkt. En daarna doe je nr++. Ja heel efficiënt, helemaal top. ⊙ ⊗

Student 11 - Writing task

T: ja jij hebt gewoon precies hetzelfde.
S: Nou de mijne heet nr, dat is praktischer
T: even kijken, private void int getallen. ⊙ Ik zie dat je hier eerst een ! had staan, maar nu staat er maar 1 =. En dat is een toewijzing.
S: ⊗ Ohh ja dubbel =
T: precies ja, heel goed. En het moet vergelijking zijn. Dus ja dat is getal++. Dit is helaas niet de correcte syntax hiervoor.
S: oke
T: weet je hoe ++ werkt?
S: uhh nee. ja dat is gewoon de waarde erbij
T: ja dat is dus het makkelijke aan ++. In plaats van getal = getal + 1, hoef je alleen maar te schrijven getal++ en je bent klaar
S: oke
T: verder helemaal goed

Student 12 - Writing task

T: je loopt door je getallen. Als getal = 2, getallen moeten kleiner zijn dan 1000. En dan doen je telkens + 2 dus dan blijft hij even. ⊙ Ho je doet hier een nieuw getal. Oh dit is het getal dat je print en dat is een ander getal dan je loopt.
S: Ja ik wist niet zeker of dat moet.
T: Ja ik zou niet weten waarom dat zou moeten. Want je doet hier gewoon hetzelfde. Ze hebben ook telkens dezelfde waarde. Alleen deze telt op hiervoor en de ander hierna.

S: $\otimes$ Ja
T: oke, verder is je functie helemaal goed. Goed bedacht

Student 13 - Writing task

T: Ja perfect, top! $\odot$ $\otimes$

A.6 Transcripts feedback session 1 (English)

T stands for Teacher, S stands for student. Before each conversation, the group the student is assigned to, is stated. The students are numbered for referencing. The symbol $\odot$ marks were the teacher knew if the task was executed correctly or not. The symbol $\otimes$ marks were the student knew if the task was executed correctly or not.

Student 1 - Parsons' puzzle

T: Oh you just wrote down the numbers
S: Yes I have done it with numbers so you have 1 2 3 4
T: Hmm that's a bit hard to check. Where is 1? Oh here and then 1 2 3 4 5 6... $\odot$
S: That sounded very doubtful
T: Yes, I expected another line first
S: $\otimes$ Oh I must have turned them around
T: Yes. It might be more convenient for next time if you just write it down
S: Yes okay
T: Because I know that it takes less time, but that makes it difficult for me to check. Thus making it more difficult for you to check.
S: Yes that is fine
T: Yes? Okay. Then it's okay.

Student 2 - Parsons' puzzle

T: Let's have a look
S: We start with the private void
T: Oh you have ..
S: we start with 6 and then we go to 7 and that is the last one
T: 6, 7, 3. Hmm I first have .. When do you initialize the number? Oh you do that afterwards. 6 7 3 is the while. Then we do the if, 10 9, yes, that's good. Then comes the else
S: and then the error comes $\otimes$
T: you are at 1 and 2. Huh hmm
S: Yes, here comes the error
T: Yes exactly. $\odot$ Do you see yourself what is wrong?
S: I see yes
T: Yes you have to print it if it is even. Then you don't have to print it for every number. That he does. So that's a bit
S: Yes
T: Furthermore, do you understand what mistake you made?
S: Yes

Student 3 - Parsons' puzzle

T: Oh yes, you wrote it down the same way. This could be better for yourself as well. Just write down the program. You already notice that it is difficult for me to check, so it must be difficult for you to check. So it is better if you can check it yourself and check whether everything went well. Yes? OK, great
S: so there are still int = 0
T: Yeah well, let's see. Oh that int, yes. He perfect! $\odot$ Well done! Was it still difficult?
S: No. $\otimes$
T: Okay, good!

Student 4 - Parsons' puzzle

T: Oh you put it like that.
S: Yes, I didn't copy it completely.
T: So we start there, then we do this, then we go here. Do we do that if .. $\odot$ No this is not right
S: Yes, I am sure it is true, ma'am. Why isn't it correct?
T: Yes, let's see, it is indeed not correct
S: Oh wait I put this on the wrong place then. $\otimes$ Okay you are right madam, it must be turned around indeed.
T: Yes exactly. Did you just not see this?
S: Yes, I did not take a good look, so it would have been better if I wrote it down, yes.
T: Okay, well. Then you also know what to do next time.
S: Yes
T: Thank you

Student 5 - Writing task

T: Let's see
S: It's a bit ugly because I didn't know what to do, that's where it starts. And there it goes further
T: okay yes. The smaller than 0, 1000. Then you do the number plus 2 and then print it
S: $\otimes$ The only mistake is that it doesn't print 0 now, but then it should print here one more time before that.
T: $\odot$ Yes exactly
S: But that didn't seem right to me
T: But then you could first print the number and then do the number + 2.
S: Yes I could, but then the good number would have gone
T: Further more. If I were you I would take a better look at the assignment next time. It says here that you have to print this. Like 6 = even.

S: so number and then string is even.
T: and you have not made a method of it. But otherwise super, great!

Student 6 - Parsons' puzzle

T: Nr = 1, while, then you do the if, then print it and then else. Hmm I miss the nr ++ ⊙
S: yes it is here right?
T: yes, but what if. Watch this while here, it checks. Suppose he is odd, then he does the if, then you print it. But this is a check if it is true
S: then it is odd
T: yes
S: ⊗ oh then you also have to do nr ++ there
T: yes exactly and then with that else, what should you do there?
S: then print
T: yes but then you also have to do nr ++ right? Otherwise it is odd once and then nothing happens. That's why there are 2 nr ++ things. Okay, thank you

Student 7 - Writing task

T: public int print numbers and then he returns the variable points. The number is less than 1000. Number is even. ⊙ You don't do a check. You start at 2. Then you print 2 is even and then number 3. And then you print it again.
S: ++ is common anyway, wait that's only 1 more
T: Yes it is not quite the correct syntax do you see that? If you do this [write example] he will do + = 1
S: And when you
T: for + = 2 you can do that rule twice, or just + = 2
S: ⊗ ooohh +=, oh yes. Do I have to write it all over again?
T: no if you understand, it's okay.
S: Yes okay. Good!

Student 8 - Parsons' puzzle

T: void evenOrDd, int no. 1000
S: yes there must be 2x nr ++, here too.
T: Yes exactly. Oh then you really made it very well! ⊙ You only forgot the nr ++.
S: ⊗ yes I only saw it once, so I didn't write it down twice
T: Yes, okay. Well done!

Student 9 - Parsons' puzzle

T: private void evenOrDd, nr = 1. Yes. Then it prints, whoops. $\odot$ Did you look at this comparison closely?
S: Probably not.
T: It says here, nr% 2. So if he is modulo 2 unequal to 0, then probably 1 comes out
S: And then it's odd, aha. $\otimes$
T: So look a little more critically at the code. But otherwise it's fine!

Student 10 - Writing task

T: method on it is also good. First initialize the variable with value. If it is less than or equal to 1000, it will continue. If he is unequal to 1, then he is 0 so he is even. Yes that's right. And that's the only thing you check. And then you do nr ++. Yes very efficient, perfect. $\odot$ $\otimes$

Student 11 - Writing task

T: yes you have exactly the same thing [as the student before you].
S: Well mine is called nr, that's more practical
T: Let's take a look, private void int numbers. $\odot$ I see you put an ! here first, but now there is only 1 =. And that is an assignment.
S: $\otimes$ Ohh yes double =
T: exactly yes, very good. And it must be comparison. So yes that is number ++. This is unfortunately not the correct syntax for this.
S: okay
T: do you know how ++ works?
S: uhh no. yes that's just the value there
T: yes that's the easy thing about ++. Instead of number = number + 1, you just have to write number ++ and you're done
S: okay
T: otherwise completely correct

Student 12 - Writing task

T: you walk through your numbers. If number = 2, numbers must be less than 1000. And then you always do + 2 so he will stay a while. $\odot$ Woah you do a new number here. Oh this is the number that you print and that is a different number than you walk.
S: Yes, I wasn't sure if I had to.
T: Yes I would not know why that should be. Because you just do the same here. They also always have the same value. Only this counts on before and the other on.

S: $\otimes$ Yes
T: okay, otherwise your position is great. Well thought out

Student 13 - Writing task

T: Yes perfect, great! $\odot$ $\otimes$

A.7 Transcripts feedback session 2 (English)

T stands for Teacher, S stands for student. Before each conversation, the group the student is assigned to, is stated. The students are numbered for referencing.

Student 1 - Parsons' puzzle

T: [Reads aloud] Yes, really good. Did you find the assignment difficult?
S: No.
T: Good.

Student 2 - Parsons' puzzle

T: Are you done as well? Lets see.
S: Yes, I only had to turn 1 thing around.
T: Oh really? It puts the rules in a random order, so you're lucky.

Student 3 - Writing task

T: Do you find it harder?
S: Yes
T: Yes you have to write it, right? That is why I have the example on the board. So that you can still look at it.
S: It's really harder.
T: It is exactly the same assignment, but then you have to write it yourself. Try again, you can do it.

Student 4 - Writing task

S: I really can't figure out what to do.
T: Have you looked at the example? If you can't figure it out then just show what you did so far. Then I will give feedback about that.
S: OK.

Student 5 - Parsons' puzzle

T: [Name] Are you also going to work on the assignment? Oh you already have it!
S: Yes, look.
T: Oh that's all right. How did it go?
S: It was pretty easy.
T: Yes, nice. Do you think you could have made it before I explained it?
S: Yes.
T: Okay, good. Do you think you could have written it yourself?

S: Yes.
T: OK.

Student 3 & 6 - Writing task

T: If you really can't figure it out, you can indicate that. Then I will help you out.
S: Yes we can't figure it out.
T: Okay then I'm going to help you. Let's see, you are both in the same place in the assignment. You have already created the method, that is very good. We have to print the characters, so you are indeed making a for loop.
S: I only did what is on the board.
T: On the board it is written that we first create a variable. We say the object type and give the variable a name. Then we say that the egg object there on the board is every egg that appears in eggList. And now we have characters. So we actually want to have every character that appears in the list. So we can take over the example almost exactly and adjust it a little bit for our situation. So you said there are strings in that list.
S: Yes.
T: What is the type of the object?
S: String.
T: Exactly. So we can write that down. Open parenthesis, String. And think of a nice name, it doesn't matter. You only need it in the for loop. Nice, good name. Then you put a colon and we need the list that we are going to walk through. That is..
S: Do you have to initialize the characters in the list?
T: Yes well you have to initialize the list first. Because you haven't done that yet.
S: umm ..
T: Shall I give an example of that? How to initialize a list?
S: Yes.
T: Okay [give an example on the board]
S: Should this be in the for loop? Or where should this be?
T: Well you have to initialize it somewhere right? And you said that you already got the list with the method. So you probably have already initialized the list before you enter the method. If you don't get the list, you can simply initialize it in the method. You can then go through the list. Because that can only be done after the initialization. Then, during the process, you can print any letter. Because you always get each letter in the for loop. And that is what you want to print, right?
S: Yes
T: So then you can print that. And then you only have to close the method. Yes?
S: OK.

T: Try this and I'll come back later.

Student 7 - Parsons' puzzle

T: Did it work completely? Did you finish the assignment?
S: No, I don't have it yet.
T: Oh you haven't done it right.
S: Oh no, I did it completely wrong.
Other student: He always has the prescribed lines, that's not fair at all.
T: There is only one such assignment left, then we will alternate it, okay?
S: This line must be back there again.
T: Yes.
S: But I actually thought I had done that.
T: Yes I don't know what happened.
S: Well then it is okay.
T: But you get it?
S: Yes, I get it.
T: OK.

Student 8 - Writing task

T: Are you figuring it out?
S: Yes a little bit.
T: You can indeed just take over from the board and adjust it to what you need. Well done!

A.8 The self-efficacy questionnaire (Dutch)

Naam:	Compleet niet waar voor mij						Heel erg waar voor mij
1. Ik geloof dat ik een excellent cijfer zal krijgen voor het programmeerdeel van dit vak	1	2	3	4	5	6	7
2. Ik ben er zeker van dat ik de meest ingewikkelde materialen over programmeren die we hebben gekregen begrijp	1	2	3	4	5	6	7
3. Ik ben er zeker van dat ik de basis concepten die mij geleerd zijn in programmeren, begrijp	1	2	3	4	5	6	7
4. Ik ben er zeker van dat ik de meest ingewikkelde concepten die mij geleerd zijn door mijn docent in programmeren, begrijp	1	2	3	4	5	6	7
5. Ik ben er zeker van dat ik het fantastische zou doen bij opgaven en toetsen over programmeren	1	2	3	4	5	6	7
6. Ik ga er vanuit dat ik het goed ga doen op het programmeerdeel van dit vak	1	2	3	4	5	6	7
7. Ik ben er zeker van dat ik de vaardigheden die mij zijn geleerd in programmeren, kan beheersen	1	2	3	4	5	6	7
8. Als ik rekening houd met de moeilijkheid van programmeren, de docent en mijn vaardigheden, denk ik dat ik het goed zal doen	1	2	3	4	5	6	7

Table A.1: The self-efficacy questionnaire in Dutch