Bachelor thesis
Computing Science

Radboud University

Exploring Python as a replacement for C++ in Imperative Programming for Computing Science at Radboud University

Author:
Huy Nguyen
s4791916

First supervisor/assessor:
Dr. P.Achten
P.Achten@cs.ru.nl

Second assessor:
Prof. Dr. E. Barendsen
E.Barendsen@cs.ru.nl

June 28, 2019

Abstract

The first programming course for computing science students at Radboud University is Imperative Programming in C++. In this thesis we explore whether Python can potentially be better than C++ for this course. To do this we reviewed the literature to gather advantages and disadvantages of using Python as the first programming language. Following this we analyzed 1783 assignments that the students handed in for various errors and problems. Some teaching assistants of the course were also interviewed. Based on this research we conclude that Python can potentially be better as it can help students reduce errors made due to syntax, improve their process of understanding data structures and spend more time on problem solving instead of learning syntax.

Contents

Chapter 1

Introduction

Teaching students how to program is not always an easy task [15, 16, 22, 39]. There are many factors that can influence students [18]. A factor that is known to influence students is their first programming language [10, 36, 38]. At Radboud University many computing science students have no prior programming experience when they start studying and thus it is important that their first programming language gives them a solid foundation to build upon in future courses. The computing science students at Radboud University start programming in C++ with the course Imperative Programming, but the top American universities have been switching to Python for their introductory programming courses [12] and teachers at UK universities have also noticed the educational benefits of Python [20]. Along with this, Python has also seen more usage in the industry [31].
With the rising popularity of Python, the goal of this thesis is to find out whether a potential change from C++ to Python in the course Imperative Programming for Computing Science at Radboud University would be a worthwhile decision. In order to achieve this goal, the research in this thesis is split into two parts. The first part will be a literature study of the effects of using Python in an introductory programming course. This course is also commonly referred to as CS1. In this part we try to answer the following research question.

- **What are the reported advantages and disadvantages of using Python in CS1 in a university setting?**

This will give us an idea about why lecturers have been switching to Python and help us look at the correct aspects when considering a switch ourselves.

The second part will be an analysis of the course Imperative Programming for Computing Science at Radboud University in which C++ is used. Here we try to answer the following two research questions.

- **Which programming concepts do our students struggle the most with?**
- **How does the code of our students adhere to certain coding standards?**

The goal behind these two questions is to have a close look at the struggles of our own students. If they are struggling with certain problems in C++ then switching to Python would only be worthwhile if they end up struggling less with these problems in Python.

Both parts of our research lead up to the main research question.

- **Could Python potentially be a more suitable first programming language for Computing Science at Radboud University?**

The structure of this thesis is as follows. In chapter 2 we go over the preliminaries. The first part of the research is in chapter 3 where we perform a literature review. In chapter 4 we describe the methodology that we used to perform the second part of our research. The results of this will be shown in chapter 5. Chapter 6 will be a discussion of the literature review and results. Chapter 7 will be related work. Lastly chapter 8 will be our conclusion and some possible future work.

Chapter 2

Preliminaries

2.1 The course Imperative Programming

The first programming course for computing science students at Radboud University is Imperative Programming in C++. This course is also taken by a variety of students from other studies as well. The main goal during this course is not to learn C++ but to use C++ as a tool to learn basic imperative programming concepts and skills. The first half of the course focuses on programming basics while the second half introduces more advanced algorithms and concepts. Students have a lecture and tutorial every week where new programming concepts are introduced and explained. In the first half of the course there is a mandatory weekly test that prepares them for the weekly assignments. This test contains questions that go over the programming concepts that were explained in the lecture. However, this test does not count towards their final grade and the only requirement is that students do this test. Furthermore, there is also a lab session with teaching assistants where they can work on the weekly assignments and ask questions if needed. The IDE that is recommended to the students is Codeblocks [4] which uses the GNU GCC compiler version 5.1.0. The lab sessions are split up based on the programming experience (beginner, experienced, advanced) of the students. Students work in pairs and hand in their solutions after the lab session. The teaching assistants will grade these assignments and provide formative feedback. The grades that can be given are fail, insufficient, sufficient and good. Students are not allowed to fail any assignment as it excludes them from taking part in the mid-term exam and end-term exam. However, the grades themselves do not count towards the final grade. On the next page you can find a summary of the weekly assignments, the background of the students and their programming experience.

Week	Assignment	Programming concepts
1	Charles the robot	Void functions, boolean operators, while loops and if-statements
2	Charles the robot part two	Parameterized functions and control structures
3	Square root algorithms	Data types and input/output
4	Easter	Functions with return values and call-by-value
5	Encryption and decryption	Pre/post conditions, working with text file input/output and call-by-reference
6	Game of life	One-dimensional and two-dimensional arrays
7	Concordances	Arrays and input/output
8	Music database	Structs and vectors
9	Music database part two	Sorting algorithms
10	Heap sort	Runtime complexity and binary search
11	Introduction to recursion	Basic recursion
12	"Pakjesavond"	Recursion with branch and bound algorithms, depth first search
13	Sokoban	Recursion with branch and bound algorithms, breadth first search
14	Quicksort	Recursion versus iteration

Table 2.1: Assignments and most important programming concepts taught every week

In weeks 1, 2, 4, 5, 6, 8 and 9 some code was already provided for the students to work further upon. These template files can be found here.[1] Week 3 and 4 also offered students the opportunity to follow a guided lab session in which they were helped step by step to create a final solution. The background and programming proficiency of the students can be found below. Note that only 192 of the 255 registered students for this course indicated their level of programming proficiency during the first lecture.

Major	**Percentage**
Computing science	70%
Physics and astronomy (mostly bachelor, few master)	8%
Mathematics	5%
Other science studies	6%
Management sciences (mostly bachelor, few master)	7%
Other studies or contract	4%

Table 2.2: Student background

Programming proficiency	**Percentage**
Beginner	40%
Experienced	55%
Advanced	5%

Table 2.3: Programming proficiency of the students

[1] https://github.com/huynguy97/Thesis-data/tree/master/template%20files

2.2 Cppcheck

Cppcheck [5] is a static analysis tool. Static analysis is when you analyze the code for potential issues without executing the code. For example, a compiler can give a warning for uninitialized variables without needing to execute the code. However, there are some issues where a compiler will not return an error or warning. This is where Cppcheck comes in as it tries to detect issues that a compiler will not find. There are different types of messages for the issues that Cppcheck finds [6]. We will only be looking at three type of issues: errors, warnings and style. The code below shows us an example of code where Cppcheck detects an error while the GCC compiler does not give an error or even a warning.

Invalid argument for abs:

```
1 while( abs( x*x - v > e) )
2 {
3     x = x - ((x*x - v) / (2*x));
4     iteration = iteration + 1;
5 }
```

The error is here is that a boolean value has been passed to the function `abs` which expects a numerical value.

2.3 C++ and Python

In this section we highlight a few relevant differences between both programming languages. Relevant as in subjects that the Imperative Programming students at Radboud University would face themselves during the course. We will first provide a little background information on both languages.

2.3.1 Background information

Bjarne Stroustrup began working on C++ in 1979 and in 1985 C++ was released [28]. The reason for creating C++ is that he wanted to create efficient system programs. Stroustrup added features to the programming language C and one of the most important features was being able to work with classes [29, 30]. C++ gradually grew in popularity and is currently still very popular [31].
Python was created in 1989 by Guido van Rossum [32, 34]. One of the reasons being that he was frustrated with C as it took "weeks to write a simple program". From the start Python was designed to be easy to learn [34] and with a focus on readability [33]. Van Rossum also took inspiration from the programming language ABC which was designed for teaching [33].

2.3.2 Some differences between C++ and Python

Starting off with one of the most notable differences, the syntax of Python allows for programs to be shorter. Even something as classic as printing hello world requires less code in Python.

Hello world in Python:

```
1 print("Hello world!")
```

Hello world in C++:

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     cout << "Hello world!";
5     return 0;
6 }
```

In Python there is also no need for a semicolon to end a statement and Python relies on indentation for grouping of statements. For instance, in C++ the body of a function is declared between curly braces and semicolons are used to end a statement. In Python, no semicolons are needed and indentation is used to indicate the body of a function. Below an example is shown with an if statement in both languages.

If statement in Python:

```
1 if x:
2     print("No semicolons!")
3     print("No curly braces!")
```

If statement in C++:

```
1 if (x){
2     cout << "Semicolons!";
3     cout << "Curly braces!";
4 }
```

Python also has an interpreter [25] and while some have created an interpreter for C++ [3], using it is not as easy and as common as the Python interpreter. The interpreter allows to interactively execute Python code.

In C++ arguments can be passed by value or by reference. In Python this works a bit different. In Python passing arguments by reference in a similar way as in C++ is not possible. Does that mean that everything in Python is passed by value? This is also not the case. In Python arguments are passed by assignment [23]. This works as follows. Everything in Python is an object [24]. Some objects are mutable and some are immutable. Passing a mutable object such as a list makes it possible for a function to manipulate this list as long as it does not try to reassign it. However, passing an immutable object such as an integer and trying to change its value in any way is not possible. Fortunately, there is a workaround for this. We can create a new variable, perform the operations we want and return this new variable. An example is shown on the next page.

Attempt to change an immutable object in Python with reassignment:

```
1 def change(number):
2     number = number + 1
3
4 x = 5
5 change(x)
6 print(x) # prints 5
```

Workaround with a new variable and return statement:

```
1 def change(number):
2     value = number + 1
3     return value
4
5 x = 5
6 x = change(x)
7 print(x) # prints 6
```

In the left function the reassignment of `number` would not change variable `x` as integers are immutable. The variable `number` in the function will refer to a new object and this will be unknown outside its scope. If we want to pass back that `x` has changed we do this creating a new variable, performing the operations we want and returning its new value. This is shown on the right. For mutable objects such as lists, we can perform any operation we want as long as do not try to reassign the list.

Changing a mutable object in Python:

```
1 def change(numbers):
2     numbers.append(2)
3     numbers = [1,2,3] # reassignment not seen outside the scope
4
5 x = [1]
6 change(x)
7 print(x) # prints [1,2]
```

We see that appending `2` changes the actual list and not just a copy of it, but when we try to reassign the list to `[1,2,3]` this change is not reflected back. Once again we have just created a new object `[1,2,3]` that is referred to by the variable `numbers` and this is unknown outside the scope of the function.

Another difference is that Python uses dynamic typing [26]. As we have seen so far there is no need to declare the type of a variable like we are used to doing in C++. As Python uses dynamic typing the type of a variable is figured out during runtime.

Chapter 3

Literature review

The literature around introductory programming education is extensive [18]. For this thesis we focus on the effects of using Python in introductory programming courses. We search for commonly reported advantages and disadvantages of Python in order to find out whether there is a general consensus on certain advantages or disadvantages of Python. We will only consider features of Python that are relevant for this thesis. For example, many authors mention the support of object-oriented programming in Python to be a positive point. However, this is irrelevant in the context of this thesis.

Already in 1999 Zelle [39] claims that Python is a "nearly ideal" candidate as the first programming language of a computer scientist. He first goes over some assumptions about CS1/CS2. The goal of these classes is not to learn a certain programming language, but to learn problem-solving, design and programming. CS1/CS2 is an introduction to the field of computer science and not an introduction to a particular programming language. It should also introduce students to the primary programming paradigms. The last assumption is summarized by him as: "Programming is hard, but we should strive to make it no harder than it needs to be.". According to Zelle, these assumptions imply that a first programming language should have simple syntax and semantics. Zelle makes a case for Python and mentions some advantages of Python. For example, the simplicity of Python allow students to have an easier time learning the language and avoiding syntax errors such as missing semicolons. Another advantage is that more interesting projects are possible thanks to the data structures in Python and the fact that Python requires less code overall in comparison with other languages. He goes on to conclude that a scripting language such as Python is thus ideal in an introductory programming course.

In 2006 Patterson-Mcneil [22] had taught CS1, CS1.5 (object-oriented programming) and CS2 (data structures) for ten years in C++ at two different places. At her current college the CS1 course is taken by computer science, math, chemistry and biology students. The CS1.5 and CS2 course are only taken by CS students. However, it should be noted that most of the students of Patterson-McNeil were "non-traditional". Students with children, a job and mostly older than those that come directly from high school. She noticed that a lot of students struggled and dropped out. Only 10% of the students who started with CS1 also finished CS2. Unsatisfied with these numbers and C++, she attended the SIGCSE 2003 conference and took part in a workshop from Zelle who was sharing his experiences of teaching with Python. Convinced by Zelle, she decided to switch to Python. This was done in three iterations over the next three years. At first Python was only introduced as a part of the CS1 course. Python was used during the first 10 weeks and students were taught C++ nearing the end of CS1. As this did not improve much and students did not understand why they needed to learn C++ at the end of the course, the author decided to switch CS1 completely to Python. Although this helped the CS1 course, it did not yield much improvement in CS1.5 and CS2. The last step was to change CS2 to Python as well and shift the CS1.5 course to a later stage in the curriculum. The simple syntax and data structures of Python made it possible for more interesting programs to be written by students in CS2. Another positive point was the interactive shell as students could interactively test parts of their code with it. For Patterson-McNeil switching both CS1 and CS2 to Python reduced the amount of students that dropped out. 40% of the students that started with CS1 also finished CS2, which is an improvement from the old 10%.

In 2000-2001 Centre College [27] switched from C++ to Python for a breadth first CS1 course taken not only by CS students but also by many other students from different majors. This is called a breadth first CS1 course as it includes many topics relevant to computer science aside from just programming, such as the internet, world wide web and social/moral issues related to computer science. However, the main focus was still programming.
Oldham in 2005 [21] describes how this switch went and why Centre College was satisfied with Python. Their CS1 course had three goals. These goals being that students should have fun in CS1, learn basic programming concepts and also learn object-oriented programming. In CS1 the simple syntax and data structures of Python made it possible for students to write interesting programs while the interactive shell also benefitted students as they could experiment more. All in all Python was a good choice in CS1. However, Oldham also mentions some disadvantages of Python. The dynamic typing in Python makes it that students have a weaker understanding of many typing concepts. Another disadvantage is that teaching the differ-

ence between call by value and call by reference was hard in Python. In his conclusion Oldham shows that although the retention and next course enrollment had not improved with Python, students and lecturers were satisfied with Python.

In 2005 Miller and Ranum [19] describe their experiences teaching two introductory programming courses to computer science students using Python. Their goal during the courses is to make sure that students focus on problem solving, algorithm development and algorithm understanding. They believe that Python is a good choice for this. They first teach students basic programming concepts in an imperative style. The simple syntax of Python allows students to quickly understand programming concepts and the interactive shell was an additional benefit for teaching and learning programming.
In their second course the focus is on data structures, a wider range of algorithms and teaching students more advanced programming features that will help them transition easily to other languages. They show various examples of why Python helps achieving these goals better than a language such as Java. For instance, they compare the implementation of insertion sort in Python and Java. Insertion sort in Java requires the creation of a class, using a method to compare values and the usage of a Java interface. Thus a lot of concepts need to be explained to a student who wants to understand insertion sort in Java. On the other hand, a Python implementation of insertion sort is much less complex. They show a few more examples and in their conclusion they also mention that their students were able to complete over twice the programming exercises than in previous years when they used Java.

Leping et al. [17] talk about how and why they decided to switch the language in their introductory programming course from Java to Python. They first conducted a small experiment in the study year 2008/2009 in which a small group of students was taught Python and the rest was taught Java. The students were mostly math and computer science students. The main reason for choosing Python was the clean and easy to read syntax. Some other mentioned reasons were the interactive shell and that Python is a good first language for future object-oriented and functional programmers. This small experiment was successful as students who used Python scored better grades. Notably, the amount of students that failed the Python course was 14% while 28% failed the Java course. Following these results, Leping et al. decided to switch their introductory programming course completely to Python for all students in the next year.

In 2014 Ateeq et al. [2] compared C++ with Python from a students perspective. Their computer science students used Python in their first semester and C++ in their second semester. At the end of the second semester, they

collected results from a two part survey conducted among the students and interviews with selected students. The first part of the survey consisted of questions regarding the difficulty of programming constructs such as the usage of variables or loops. Students perceived looping to be easier in Python, but algorithmic design and functions were considered to be harder in Python. The authors say that this can be justified by the fact that most students were new to programming when they used Python. Another reason could be that students were making such simple programs that they failed to understand why functions were needed at all. The second part of the survey consisted of questions about certain features/aspects such as ease to program in, flexibility and simplicity. The results showed that students thought Python was better in all of these aspects.
In the interviews with students, Ateeq et al. also asked students to choose which programming language they considered to be better for a first programming language. 60% chose Python as the better choice over C++. Students considered Python to be easier and having a simpler syntax. The students that chose for C++ were mostly those that already had some experience with C++. Students were also asked about their perceptions of a first programming language. Most students answered that they expected a first programming language to be easy. For which Python is a suitable candidate. Another question that was asked was related to problem solving. Should students learn problem solving independently from learning a language? 75% of the students thought that problem solving and learning a language should be taught together. From this the authors once again conclude that Python is a good choice for this as Python is easy to learn.

However, not everyone has had positive experiences with Python. In 2013 Hunt [13] switched from Java to Python for his CS1 course, but in 2015 he decided to go back to Java. The main issue for Hunt was the lack of real arrays. As his course involved working with images that were represented using arrays this was a problem. Lists were not a viable substitute and Hunt tried using the numpy library, which has arrays, as a workaround, but it was not an adequate replacement as well. Another issue was that in the following Java CS2 course students had to spend too much time adjusting to Java. Hunt concludes by saying that the loop style of Python was also a problem for him. The style of Python loops is not similar to many top languages that students are likely to use later on. All of these issues made him switch back to Java.

Most of the previously mentioned papers lack quantitative data and when quantitative data is used it is mostly in the form of student grades or questionnaires [16, 37]. For example, Koulouri et al. [16] discuss that the lack of quantitative data and analysis is a bigger issue in computing education research. In their own research in 2014, they therefore put a strong emphasis on quantitatively evaluating their results. Koulouri et al. [16] researched

the effects of three factors in teaching programming. One of them being first programming language choice. They compared a group of students who took a CS1 Java course to those that took a CS1 Python course in order to find out whether the choice of first programming language matters. They analyzed the assignments that the students wrote for the usage of keywords (if, for, while, etc). These assignments came from a 1 hour laboratory test that was conducted after ten weeks of teaching. As the assignments of the laboratory test were created in such a way that a perfect solution required using a keyword, it is not possible to avoid using certain keywords. Therefore the usage of keywords could be indicative of students understanding them and using them correctly. Using the correct keyword too many times to solve an assignment would also not increase the count. For example, if a solution required just one if statement, using three if statements would still be counted as one. They also looked at the amount of bugs in the assignments. The results showed that Python students used significantly more keywords and had less bugs in comparison to the Java students. For instance, Python students had 24% less bugs and also used 37% more loops than the students that programmed in Java. In their discussion they attribute this to Python having a simpler syntax than Java. For example, the increased usage of loops when students programmed in Python could be explained by the fact that loops are easier to use in Python thus leading to better understanding of the underlying concept. They conclude by saying that programming language choice does matter and it seems that more complex languages might affect students negatively.

In similar fashion Jayal et al. [15] compared their object-oriented CS1 Java course taught in 2008-2009 to their imperative CS1 Python course taught in 2009-2010. In both courses students were tested after ten weeks on their ability to write a working program. Jayal et al. compared the programs of Java students with those of the Python students. They compared the frequency of keywords and bugs in programs. As using the correct keywords and writing programs without bugs were part of the marking criteria for their final grade, students could not avoid using keywords to create a correct solution or else it would lower their grade. Their results showed that programs written by students in Python used more keywords. For example, Python students used if statements in 36% of their assignments while in Java this was only 23%. Python students also wrote code with less bugs. Only 28% of the assignments contained bugs while in Java this was 57%. Students also scored better grades in the Python course. Notably the amount of failures was a lot less. In Python this was only 11% while in Java this was 50%.

Wainer and Xavier [37] compared their CS1 C course to their CS1 Python course. This CS1 course is required for science, computer engineering, math, statistics, and all engineering majors. A total of 391 students took either

the C or Python course in the year 2015 or 2016. The authors believed that they only tested a few aspects of Python as their Python course was a "C-based CS1 course taught in a different language". Instead of building the Python course up from the ground, the authors took their existing CS1 C course and tried teaching the same subjects as good as possible in Python. Therefore the only aspects of Python that they tested were the simpler syntax, dynamic typing, Python lists, interactive environment and the lack of call by reference.
In order to measure whether the CS1 Python students performed better than the CS1 C students the authors measured the following: dropout rate, failure rate, midterm grades, final exam grades, proportion of completed weekly assignments and number of submissions per completed assignment. Students in the CS1 Python course performed better in almost all of these measurements. Only the dropout rate remained the same in Python and although the failure rate was lower in Python, it was not statistically significant. From their results we can for example see that Python students scored an average midterm grade of 7.08 which is higher than the 6.41 for C students. The final exam grade was also higher, 7.55 for Python students compared to a 6.72 for C students. The authors conclude that using Python resulted in a better outcome for their students.

Most of the previously mentioned papers looked at switching to Python for CS1/CS2 and only measuring the impact of this in CS1/CS2. Almost all of the papers that we have seen have not looked at how switching to Python for CS1/CS2 impacts the rest of the curriculum. Enbody et al. [8, 9] switched their CS1 course from C++ to Python in 2007 and went on to measure student performances in future courses. They compared students who took CS1 in Python with those that took CS1 in C++. The first programming course that followed after CS1 was their CS2 course that was taught in C++. Their results showed that the students who followed CS1 in Python did not perform better or worse than the students that followed CS1 in C++. In a subsequent research they went on to measure whether the change to Python would also affect students in other courses. Specifically, they measured the performances of students in a C-based computer organization course, a C++ object-oriented course and a C++ algorithms/data structures course. They once again conclude that students who took CS1 in Python did not score better or worse than students who took CS1 in C++.

3.1 Summary

Now that we have discussed some papers, the following two tables will provide a short summary on the mentioned advantages and disadvantages.

Advantage	**Mentioned by**
Simple syntax	[22, 39, 2, 17, 21, 19, 16, 37, 9, 15]
Data structures	[22, 39, 21, 19, 37, 9]
More interesting programs	[22, 39, 21, 9]
Interpreter	[2, 22, 17, 21, 19, 37]
Dynamic typing	[2, 19, 39]

Disadvantage	**Mentioned by**
Dynamic typing	[2, 21]
No call by value/reference	[21]
Lack of real arrays	[13]

For our research we draw inspiration from the literature. For instance, we see that a lot of researchers mention the simple syntax of Python as an advantage. Thus we will try to find out whether our students struggled with syntactical issues. Some authors also pointed out that the Python data structures were an advantage. This is something that we will also try to address.

Chapter 4

Methods

Our research consisted of two parts. In the first part we looked at the existing literature and searched for commonly reported advantages and disadvantages of using Python in an introductory programming course. This has already been discussed in chapter 3. The second part of our research was to analyze the course Imperative Programming for Computing Science at Radboud University. Two research questions were in place for this. The first research question was:

- **Which programming concepts do our students struggle the most with?**

In order to answer this question we will be looking at the weekly assignments that the students handed in. This consists of 1783 assignments from the study year 2018-2019 which have been fully anonymized and can be found here. [1] In order to obtain relevant data, the assignments are analyzed as follows.

- Using the GNU GCC compiler version 5.1.0 we first compile the assignments and collect compiler warnings from assignments that compiled correctly. We use the compiler flags `-Wall -Wextra -Wswitch-default -Wswitch-enum -Wshadow -Wlogical-op -Wuseless-cast` to collect warnings. A more in depth explanation can be found in Appendix A.2. The results can be found in section 5.2.
- From assignments that failed to compile we only collect the compiler errors. The results can be found in section 5.2 as well.
- Using Cppcheck we collect errors, warnings and style issues of assignments that compiled correctly. These result can be found in section 5.3.

[1] https://github.com/huynguy97/Thesis-data/tree/master

We only collect compiler warnings and Cppcheck problems from the assignments that compiled correctly as assignments that failed to compile often gave weird and incorrect outputs. The reason to collect more than just errors is that we can often get an indication of what students struggle with from other types of messages as well. For example, the compiler can give a warning when a non-void function misses a return statement. This indicates that a student does not understand a non-void function correctly.
However, collecting all of this does not show us what students were struggling with when they were working on the assignments. In our analysis we only work with their final solutions. Students might have struggled for hours with a certain issue but eventually they get it correct and hand in completely correct code. To get an idea of what students were struggling with during the assignments we also interviewed some of the teaching assistants that were involved with this iteration of the course. The teaching assistants have a better idea of the struggles of students because they are the ones who answer questions from students during the lab sessions and also grade the assignments. A group interview was conducted with four of the eight teaching assistants. Following this interview a questionnaire was filled in by these teaching assistants. The questionnaire was based on what we found in the literature. For example, one of the advantages of Python seems to be the simple syntax. So we asked the teaching assistants whether they noticed students struggling with C++ syntax. The questionnaire can be found in section 5.5.2 along with the results.
The group interview with the teaching assistants was conducted as follows.

- The teaching assistants were asked to rank the assignments based on difficulty.
- The teaching assistants discuss the reasons behind the difficulty of the assignments.
- The teaching assistants were shown some results of this thesis' research to discuss and look at.
- The teaching assistants were asked to fill in the questionnaire following this interview. This questionnaire consisted of 15 statements and the teaching assistants had to express how much they agreed or disagreed with these statements.

The results of this can be found in section 5.5.

Our second research question was:

- **How does the code of our students adhere to certain coding standards?**

For this we once again analyze the code. This time we make use of the code analyzer Bettercodehub, which is developed based on the book "Building

Maintainable Software: Ten Guidelines for Future-Proof Code" [35]. The choice for this tool comes from the fact that the author works at Radboud University and Bettercodehub is used in a future course as a guideline for students. While students are expected to write clean and concise code during the course Imperative Programming, this is not strictly enforced. However, it is still useful to see whether the assignments and programming language lend to writing code that is readable and simple. Any bad habit that is seen among the students can potentially be addressed in an early stage before it continues in a future programming course. Out of the ten guidelines we will only be considering the following guidelines [35]. Other guidelines are not relevant and that is explained in Appendix A.7.

1. **Write short units of code.**
   Functions should be kept within fifteen lines.
2. **Write simple units of code.**
   The amount of branch points in a function should be at most four.
3. **Keep unit interfaces small.**
   Functions should not contain more than two parameters.

These three guidelines are appropriate for this thesis as following these guidelines will lead to readable and easy to understand code. Which is exactly what you want in a first programming course where the problems the students need to solve are typically not that complex yet and should not require very complicated code. Thus checking for these guidelines will tell us whether the students actually write simple and understandable code. This benefits not only the students themselves, but also the teaching assistants who grade and help the students. The results of the Bettercodehub analysis can be found in section 5.4.

Chapter 5

Results

As listing the individual results of all fourteen weeks is too extensive we will only provide a summary of the results. See the Appendix for all details. [1] The summary of the results will be presented as follows. Firstly, we will show the number of assignments that were handed in and the corresponding grades. Afterwards the compiler errors and warnings are presented. Cppcheck errors, style issues and warnings come after this. Followed by a summary of the Bettercodehub results. Lastly, the outcome of the interview with the teaching assistants will be shown.

[1] Clicking the pictures in section 5.2 and 5.3 will take you directly to their Appendix section.

5.1 Assignments and grades

There were a total of 1783 assignments that were handed in. 1641 assignments compiled correctly and 142 assignments failed to compile.

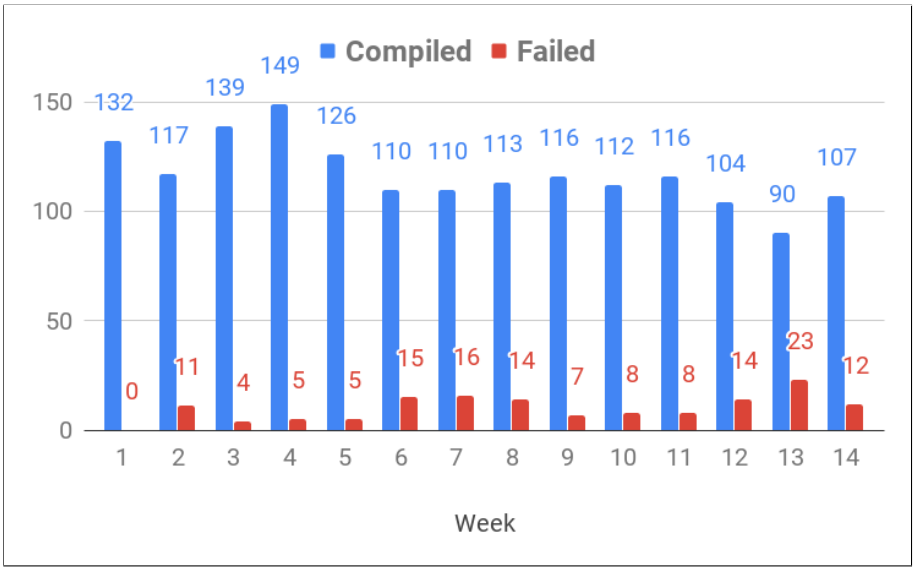


Figure 5.1: Assignments

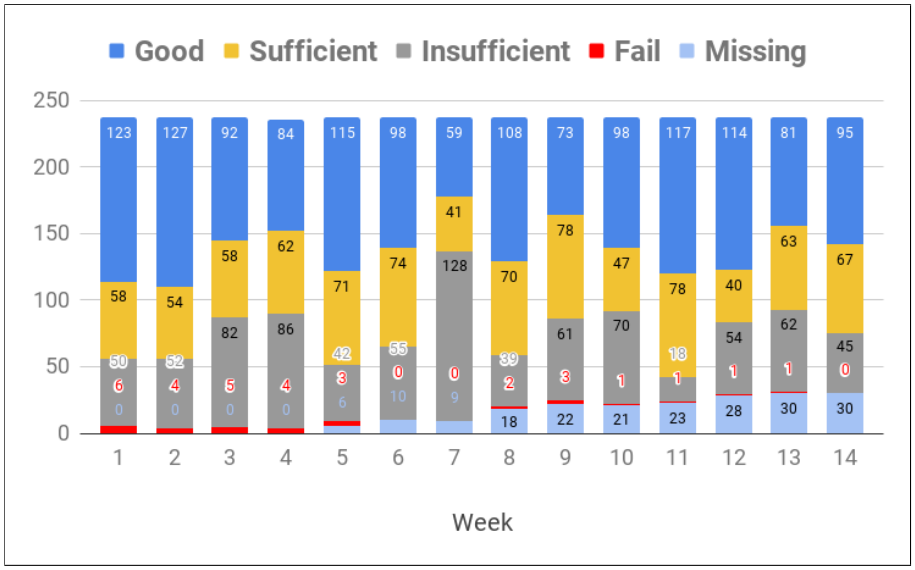


Figure 5.2: Grades

Note that 53 students participated in the guided lab session of week 3 and 64 in week 4. They were graded with an insufficient for administration reasons.

5.2 Compiler errors and warnings

There were 142 assignments that failed to compile. Multiple instances of the same error in an assignment were only counted once. For example, if an assignment could not compile due to multiple `'...' was not declared in this scope` errors it is only counted as one instance of this error. This is to combat errors that might have occurred many times in only a few files to overwhelm the numbers and make it look like this error was common among students.

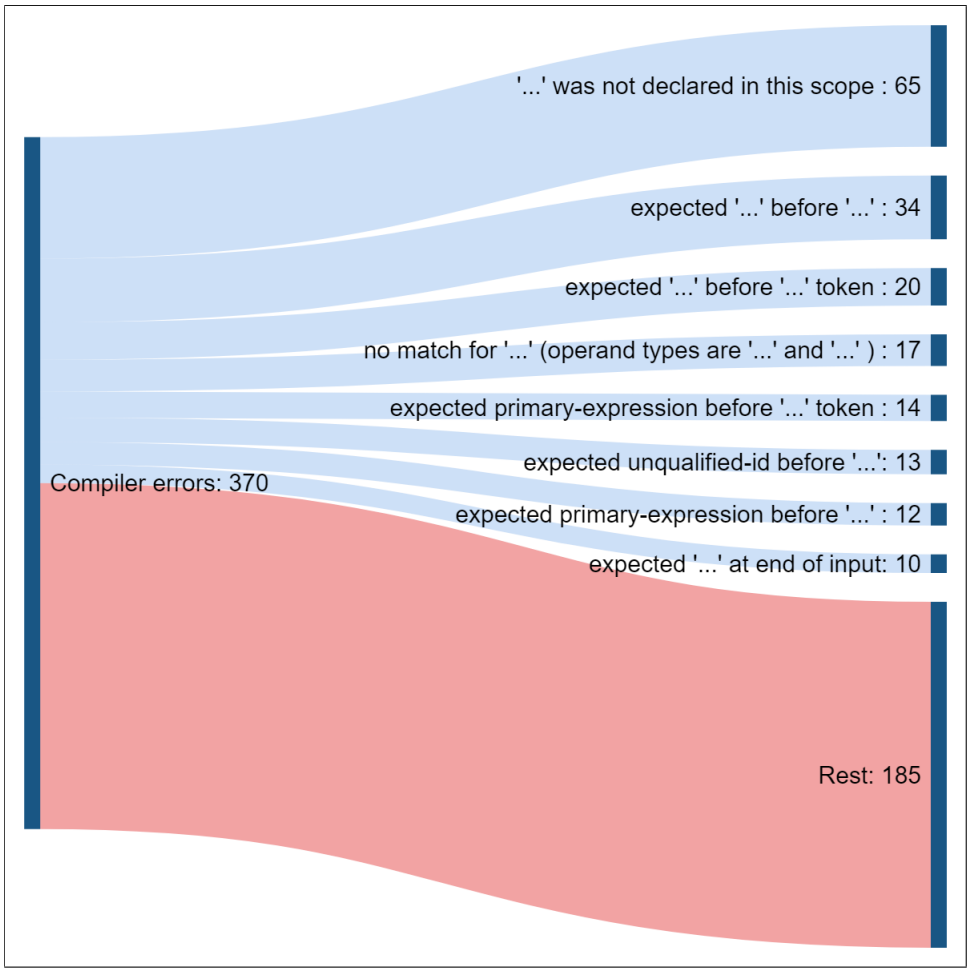


Figure 5.3: Compiler errors

The most common error is `... was not declared in this scope` which happens when a function or variable has been used that does not exist in this scope. Errors that start with `expected ...` are mostly errors due to missing certain characters such as semicolons or using them incorrectly.

There were 1641 assignments that compiled correctly. 1110 files contained at least one warning and a total of 2303 warnings was found. Multiple warnings of the same type in an assignment were disregarded. Note that in week 9 where students received template code to work further upon, the template code already contained one `Wswitch-enum` warning and two `Wshadow` warnings.

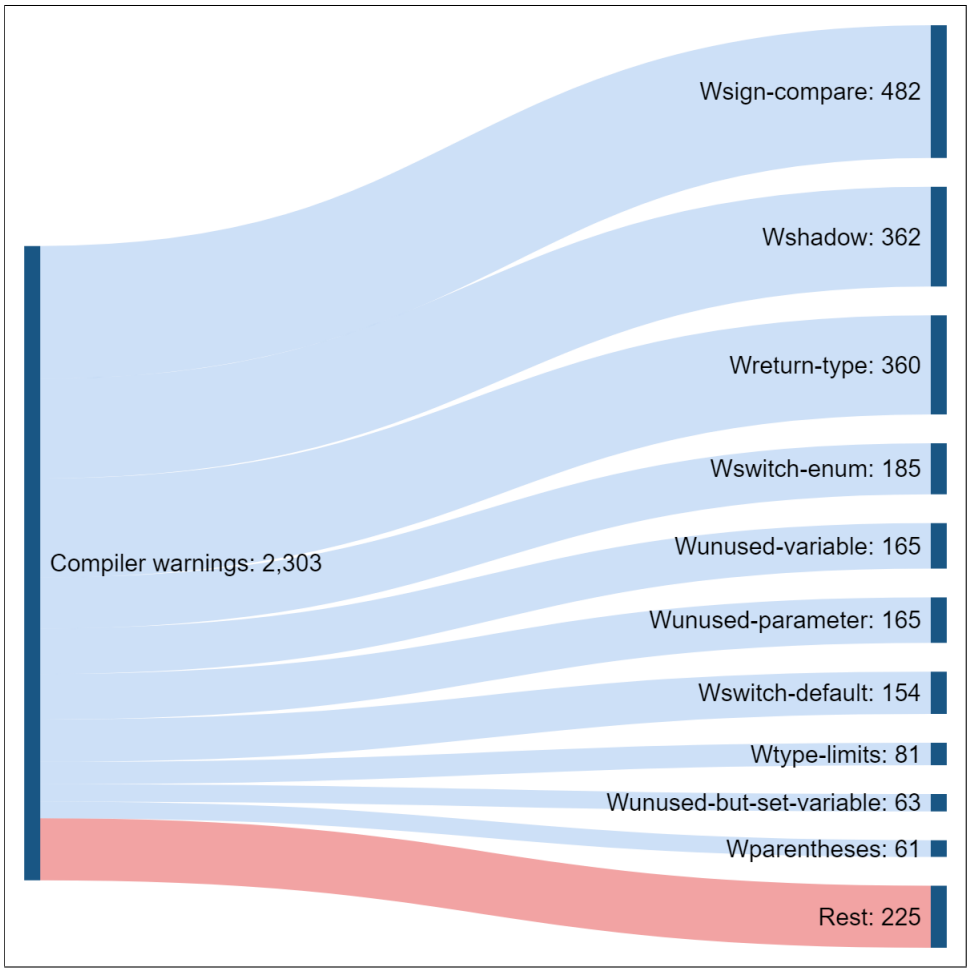


Figure 5.4: Compiler warnings

The most common warning is `Wsign-compare` which is when a signed and unsigned value is compared. This can cause incorrect results. The `Wshadow` warning is for example when a local variable is declared with the same name as a global one. `Wreturn-type` happens often when a return statement has been forgotten. Other warnings that are related to `Wunused-...` is due to not using certain parameters/variables that have been declared.

5.3 Cppcheck errors, warnings and style issues

When running Cppcheck over the 1641 assignments that compiled, 434 Cppcheck errors were found in 420 files. Remember that these are errors that are possibly not found by the compiler during compilation, but could affect the code. Once again multiple instances of the same error in a file are counted once. The error `uninitStructMember` could be disregarded as students were not instructed to initialize all struct members nor did it give any problems for the solution of the assignments. It is kept on the chart for completeness reasons.

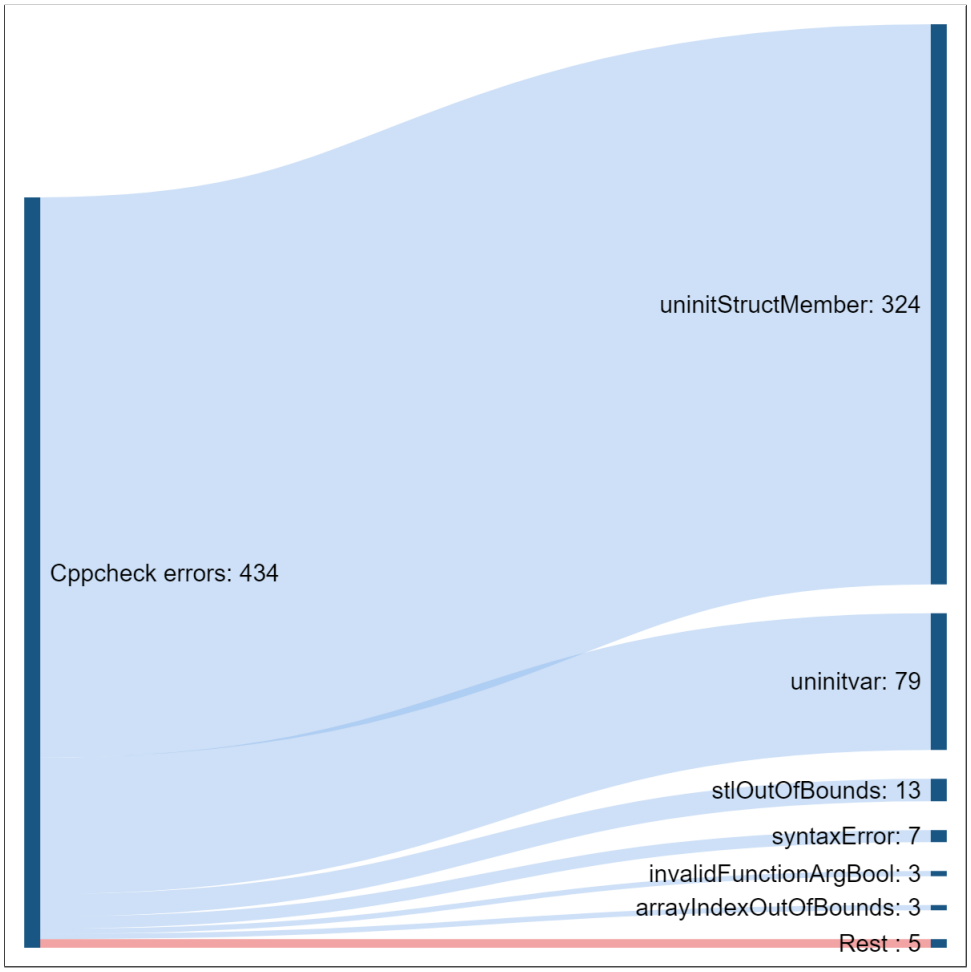


Figure 5.5: Cppcheck errors

The second most common error is `uninitvar` is due to using a variable that has not been initialized yet. It should be clarified that something that is commonly done by students such as the code below is not considered to be an error.

Not a uninitvar error:

```
1 int x;
2 cout << "Enter your number";
3 cin >> x;
```

Despite variable `x` being used before it is initialized, Cppcheck can understand the context and it is not considered to be an error. The errors `stlOutOfBounds` and `arrayIndexOutOfBounds` are out of bounds errors for data structures from the standard template library (stl) and for arrays. The error `syntaxError` is a collection of various syntax errors. An example being forgetting to declare the type of a function.

Example of a syntax error:

```
1 test(){
2     return 5;
3 }
4
5 int main()
6 {
7     test();
8     return 0;
9 }
```

The error `invalidFunctionArgBool` is when a boolean value has been passed to a function that asks for something else. Such as our example with `abs` in the preliminaries.

Invalid argument for abs:

```
1 while( abs( x*x - v > e) )
2 {
3     x = x - ((x*x - v) / (2*x));
4     iteration = iteration + 1;
5 }
```

Out of the 1641 assignments that compiled, Cppcheck found 78 warnings in 74 files.

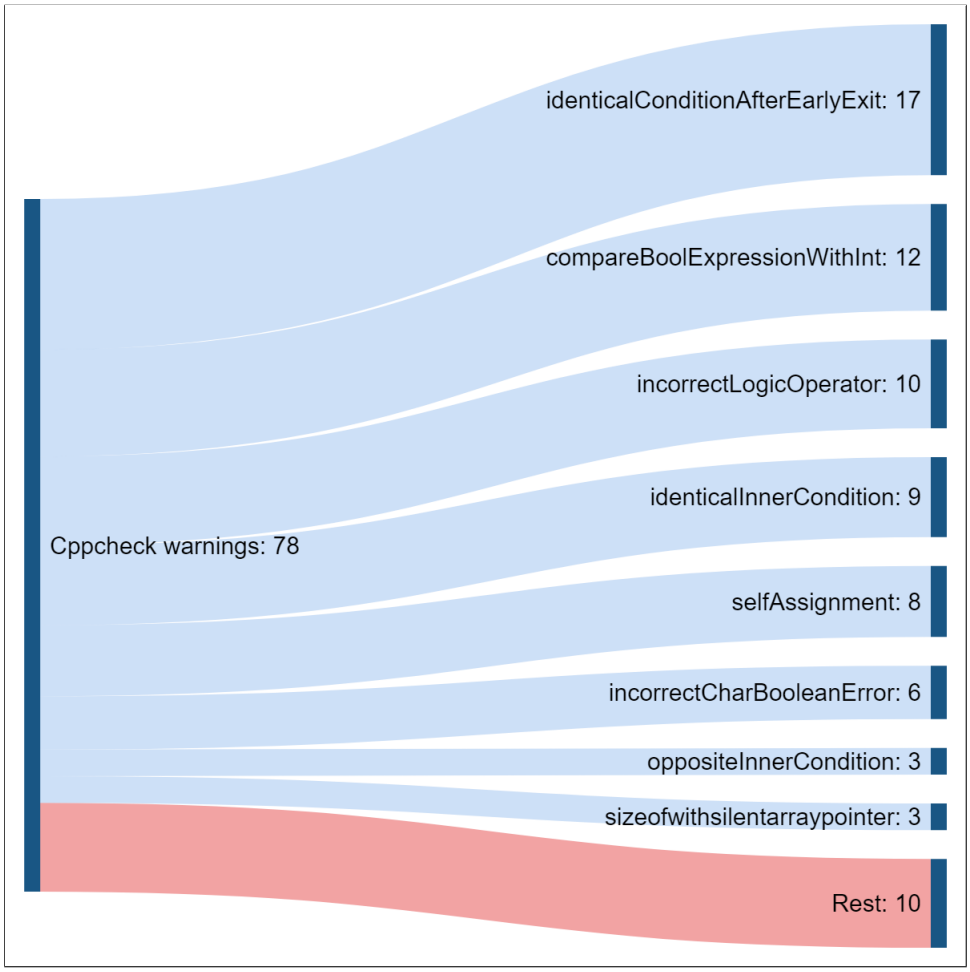


Figure 5.6: Cppcheck warnings

The warning `identicalConditionAfterEarlyExit` happens when a return statement of a function is equal to some other condition in the same function. An example of this is shown in Appendix A.6. The warning `incorrectLogicOperator` happens when both sides of a conjunction or disjunction evaluate to the same value. `identicalInnerCondition` and `oppositeInnerCondition` are warnings for control structures that have weird conditions. Either they are the same or they are the opposite of each other and lead to dead code. An example of this is also shown in Appendix A.6.

Cppcheck also found 910 files that contained 1208 style issue. The reason why `catchExceptionByValue` is on top, comes from the fact that the template code of week 1 and 2 already contained this style issue. The provided code of week 9 also contained one `shadowFunction` issue. Once again these issues are kept here for completeness reasons.

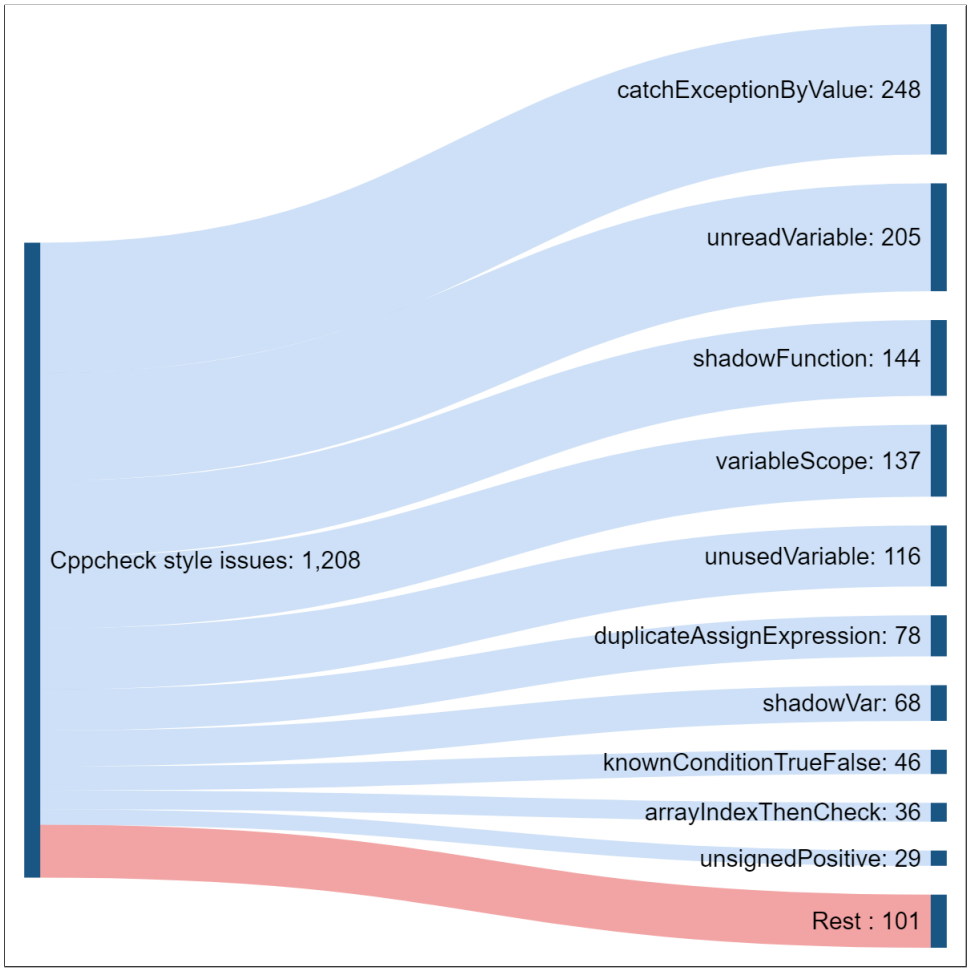


Figure 5.7: Cppcheck style issues

The style issue `unreadVariable` indicates that a variable has been assigned a value, but this value is never used anymore. The style issue `variableScope` occurs when the scope of a variable can be reduced. The style issue `duplicateAssignExpression` is for example when two variables in the same function are assigned the same value. Cppcheck wants to make sure that this is really what the programmer wanted to write. The style issue `knownConditionTrueFalse` is when a condition is checked that is always true or false. An array style issue `arrayIndexThenCheck` is when an array index might have been accessed before checking if it is within bounds.

5.4 Bettercodehub guidelines

5.4.1 Guideline: Write short units of code

The first guideline is "write short units of code". Functions should not contain more than 15 lines of code. Empty lines and comments are not counted by Bettercodehub.
The next chart shows the results of this guideline for all weeks. The chart shows the percentage of the code for that week that adheres to this guideline. For example, it shows that in week 6 approximately 20% of the code is part of a function that contains 15 or less lines. Empty lines and lines with only comments are not counted.

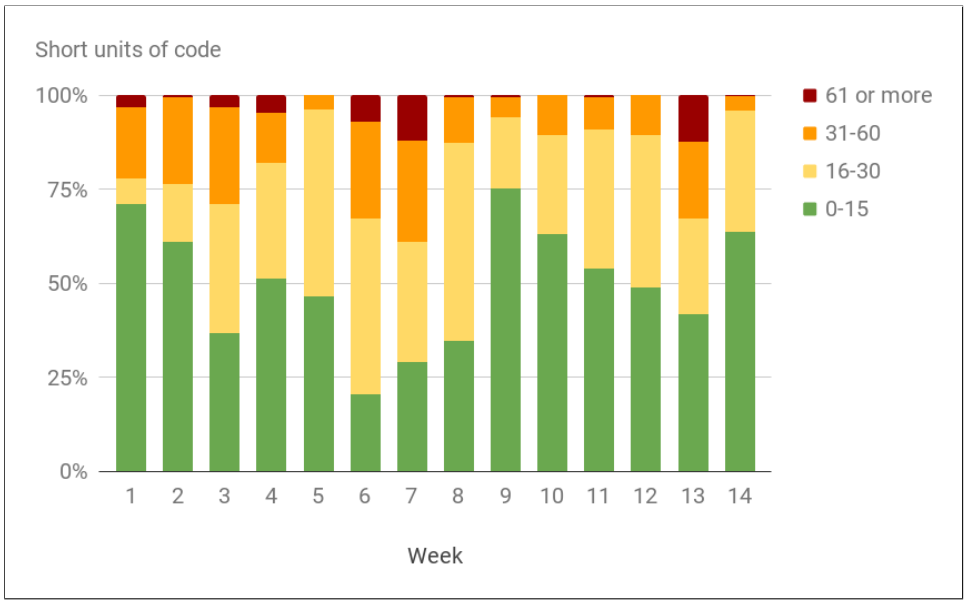


Figure 5.8: Write short units of code

5.4.2 Guideline: Write simple units of code

This chart shows the results for the guideline “write simple units of code”. This means that a function should not contain more than 4 branch points. For example, if we look at week 6, 7 and week 13 we see the majority of the code does not comply to this guideline.

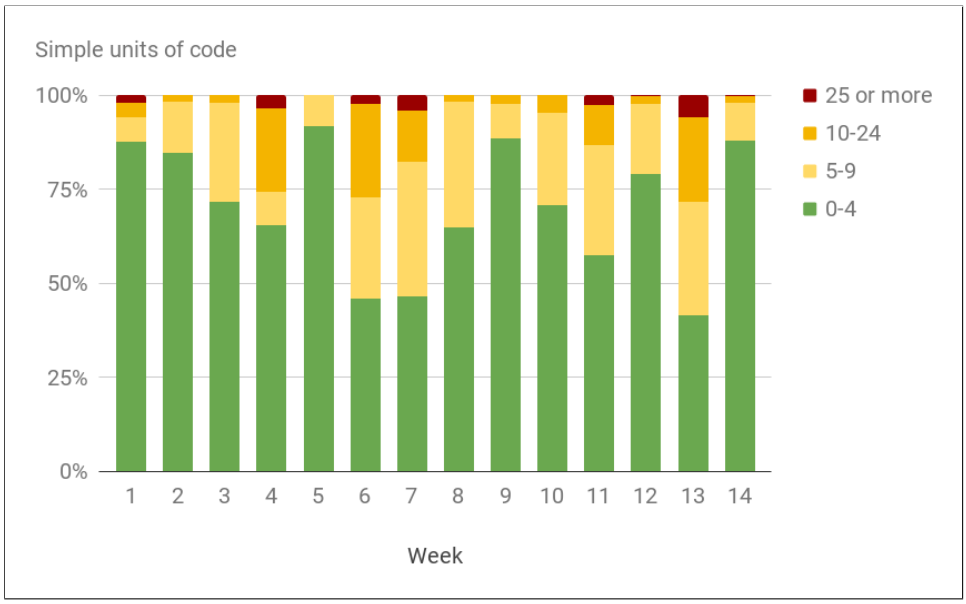


Figure 5.9: Write simple units of code

5.4.3 Guideline: Keep unit interfaces small

The third guideline is "keep unit interfaces small". This means that functions should not contain more than two parameters. Starting from week 2 students were using parameters in their own functions.

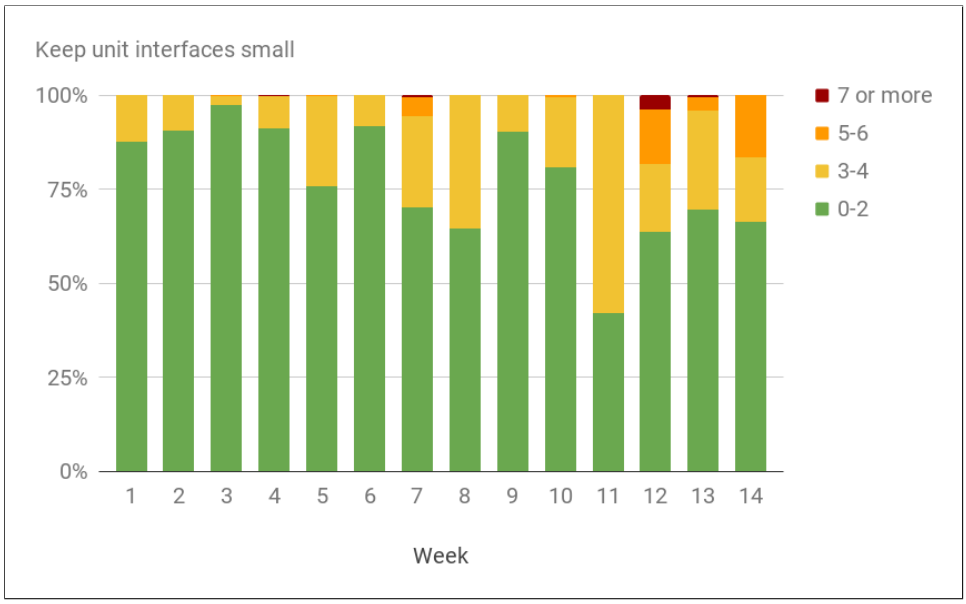


Figure 5.10: Keep unit interfaces small

5.5 Interview with teaching assistants

5.5.1 Ranking the assignments based on difficulty

The teaching assistants were asked to rank the assignments based on difficulty. A split was made between the first half and second half of the course. This means that the teaching assistants first ranked the assignments of the first half of the course and afterwards ranked the assignments of the second half of the course. The complete audio recording is available on request.

Ranking the first half of the course from hardest to easiest:

Assignment 7 Concordances: considered to be the hardest assignment. Students notably struggled with C style strings and reading the text files into data structures. Problem solving was the main issue here and syntactical issues were less prevalent.

Assignment 6 Game of life: plagued by off by one errors when working with arrays. One teaching assistant said that it was really noticeable which students started programming without really reading the assignment thoroughly and those that spent time on reading and understanding the assignment.

Assignment 4 Easter: a lot of students hardcoded the various cases. Students struggled with modulo calculations as well. When asking the teaching assistants if they thought problem solving or working with the C++ syntax was the bigger problem most teaching assistants considered problem solving to be the main issue. One teaching assistants, who was part of the beginners group, answered syntax issues as he received a lot of debug and syntax related questions.

Assignment 3 Square root algorithms: issues with creating and using functions. One teaching assistants pointed out that they were still too used to how functions work in math instead of how they were used in programming. As this was the first assignment where students were not given any code and had to start from scratch the teaching assistants also got a lot of IDE related questions. It was a steep step for some to go from working with provided code to basically building up the code yourself. One teaching assistant pointed out that some students did not understand how passing parameters to functions worked. They kept passing parameters to functions that had similar names. For instance, a function that had a parameter named "epsilon" was passed a variable with the name "epsilon" as well. Students did not understand that this was not necessary.

Assignment 5 Encryption and decryption: Some students did not understand how to work with the provided code. Working with ifstream/ofstream objects was also hard. The teaching assistants said that this was also an

issue in all assignments that involved working with these objects.

Assignment 2 Charles the robot part two: No real syntax issues. Mostly problem solving issues, but the assignment was fairly easy.

Assignment 1 Charles the robot: No real syntax issues once again. Assignment was easy and one teaching assistants received mostly questions about the IDE.

Ranking the second half of the course from hardest to easiest

Assignment 13 Sokoban: No real syntax issues or issues with data structures. Students were mostly struggling with problem solving and this assignment was pretty big as well.

Assignment 12 "Pakjesavond": Students struggled with a more advanced recursion assignment. Struggles with reading in files were only found among students that struggled with them in previous weeks. Two teaching assistants pointed out that some students just copy pasted their input/output handling code from previous weeks without really understanding them and this caused issues.

Assignment 9 Music database part two: Operator overloading was a "hell" as some teaching assistants pointed out. Students also had to implement sorting algorithm using vectors, but working with vectors was not a real issue. Some students had issues with scoping due to global variables.

Assignment 8 Music database: Reading in data from files into a struct was difficult. Understanding vectors also took some time. One teaching assistants pointed out that some students did not understand why vectors were needed in the first place. Call by value and call by reference was also a difficulty.

Assignment 10 Heap sort: Array index out of bounds issues were frequent. It took students some time but eventually they managed to figure it out.

Assignment 11 Introduction to recursion: There were not many problems with this assignment. Syntactical problems were also not common.

Assignment 14 Quicksort: Considered to be very easy. Teaching assistants pointed out that this was not hard at all. A majority of the solution was already given away in the lecture slides.

5.5.2 Results questionnaire

On the next pages you will find the results of the questionnaire. Unfortunately only three out of the four interviewed teaching assistants filled in the questionnaire. There were 15 statements and the teaching assistants had to indicate how much they agreed or disagreed with the statements.

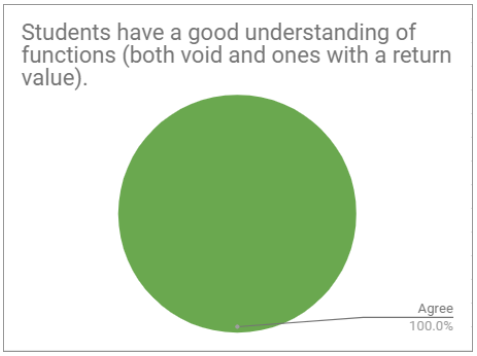


Figure 5.11: Statement 1

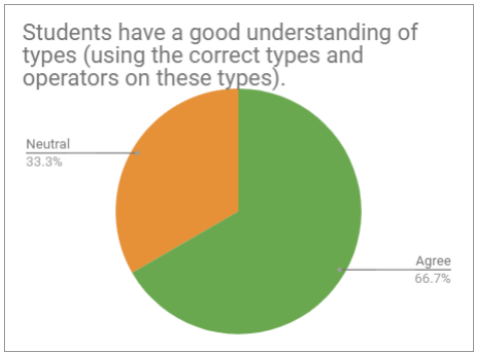


Figure 5.12: Statement 2

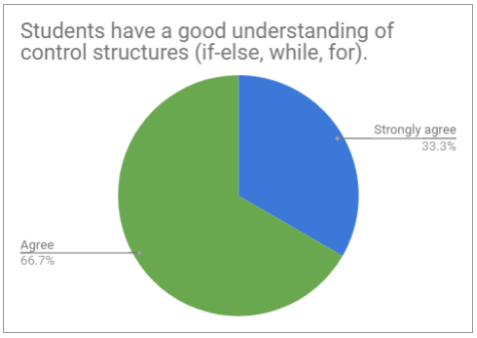


Figure 5.13: Statement 3

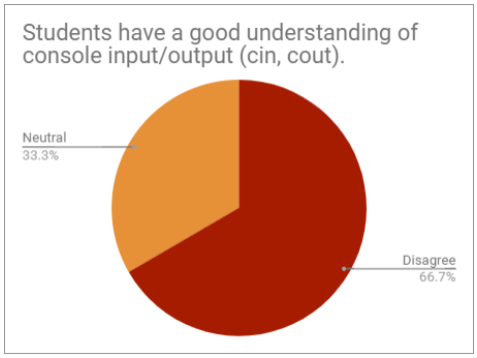


Figure 5.14: Statement 4

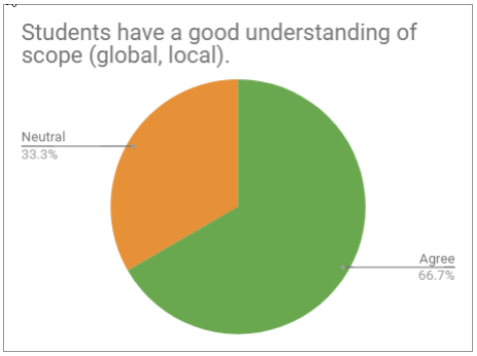


Figure 5.15: Statement 5

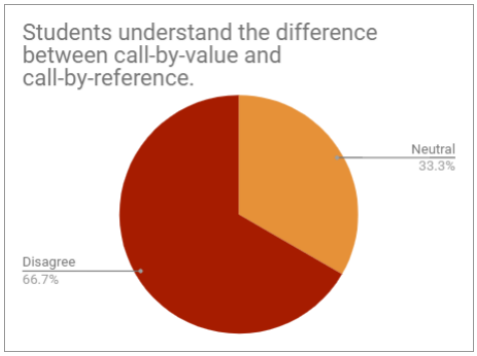


Figure 5.16: Statement 6

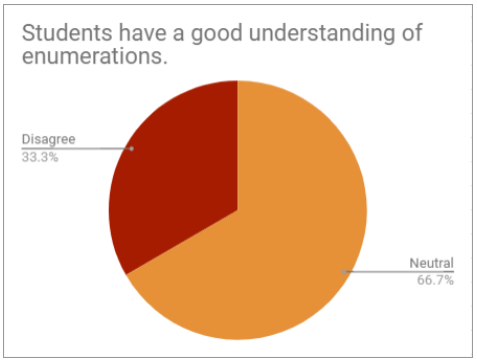


Figure 5.17: Statement 7

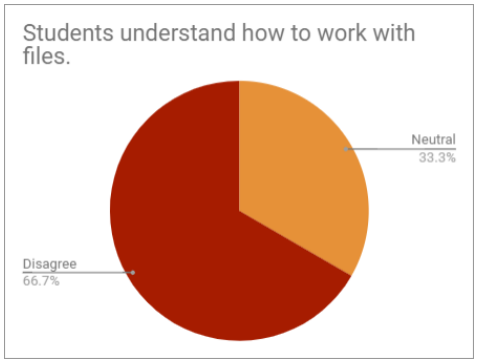


Figure 5.18: Statement 8

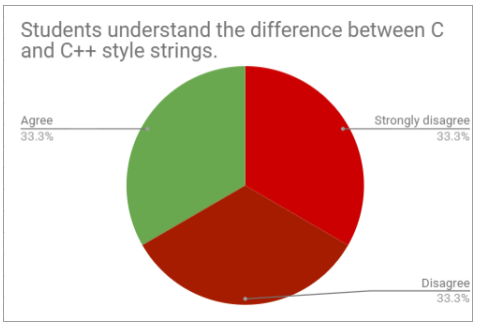


Figure 5.19: Statement 9

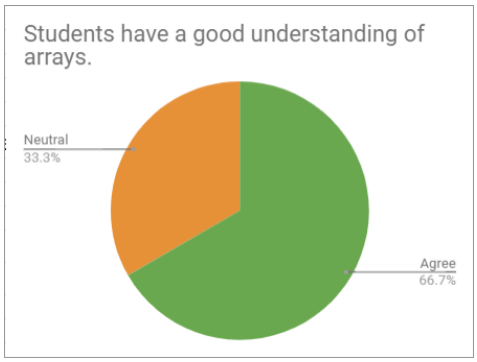


Figure 5.20: Statement 10

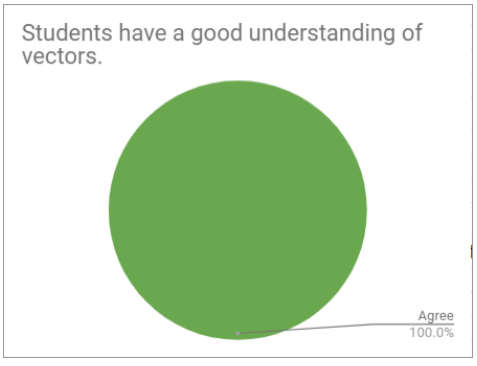


Figure 5.21: Statement 11

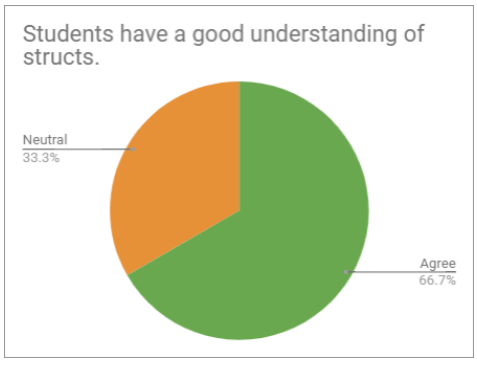


Figure 5.22: Statement 12

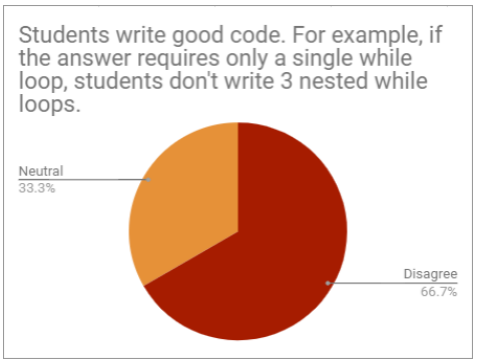


Figure 5.23: Statement 13

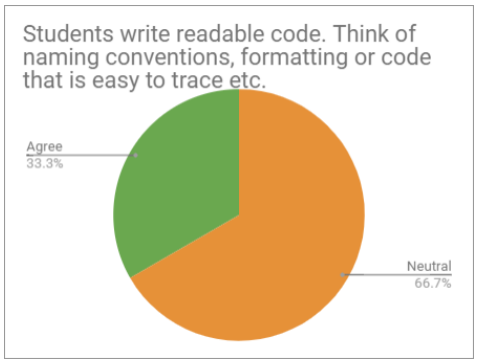


Figure 5.24: Statement 14

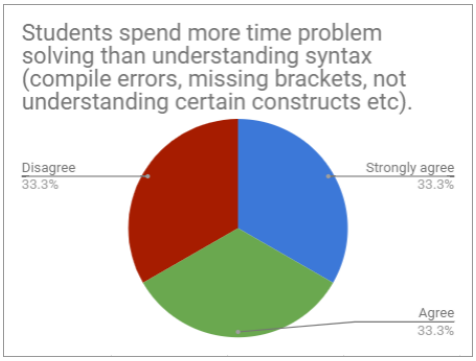


Figure 5.25: Statement 15

Chapter 6

Discussion

Our main research question was:

- **Could Python potentially be a more suitable first programming language for Computing Science at Radboud University?**

In our discussion we are mostly interested in whether Python could suit the goals of the course Imperative Programming and if Python would clearly bring any advantages in comparison with C++.

6.1 Advantage and disadvantages that we found in the literature

Advantage	**Mentioned by**
Simple syntax	[22, 39, 2, 17, 21, 19, 16, 37, 9, 15]
Data structures	[22, 39, 21, 19, 37, 9]
More interesting programs	[22, 39, 21, 9]
Interpreter	[22, 2, 17, 21, 19, 37]
Dynamic typing	[2, 19, 39]

Disadvantage	**Mentioned by**
Dynamic typing	[2, 21]
No call by value/reference	[21]
Lack of real arrays	[13]

These were the advantages and disadvantages that we found in the literature.

The most commonly mentioned Python advantage is the syntax of Python. As said before, the main goal of the course Imperative Programming for Computing Science at Radboud University is to teach students imperative programming concepts and skills. The programming language that is used for this is less of a concern. If Python can support the goals of the course while making it easier for students to program, it would be a viable choice. If students spend less time on learning the syntax of a language, they can spend more time on problem solving and focusing on the programming concepts. Another mentioned advantage is the data structures of Python. During the course Imperative Programming various data structures are introduced and used by students to solve problems. Many lecturers have seen no problems switching to Python and considered the data structures of Python to be an advantage. This would fit with the goals of the course as well. For example, students learn about arrays and vectors so they can solve certain problems. Knowledge of arrays and vectors is also a learning goal of the course. So lists in Python are a viable alternative to teach students the same problem solving skills, but they will not have the knowledge of arrays and vectors.

However, we have to consider some other disadvantages of Python as well. Some researchers have already reported that dynamic typing is not necessarily an advantage. Students will lack understanding of types and Oldham [21] already talked about how they had to address this in a follow up course. Oldham also mentioned as the only one that the difference between call by value and call by reference is hard to explain when programming in Python. Once again this had to be addressed later on.
Hunt [13] mentions that the lack of real arrays was an issue for him. Lists and numpy arrays were both not an adequate replacement. However, Hunt had a very specific way of working with arrays in his course that caused this issue. He was also the only one to mention this disadvantage while many others have considered the data structures (including lists) of Python to be an advantage. We already mentioned that the lack of arrays would not hinder teaching the problem solving skills we want to teach, but it would not give students the knowledge of arrays that we want them to have. These are some negatives and although they are not widely reported we should keep them in the back of our mind.

6.2 Compilation errors and warnings

There were 142 assignments that did not compile. The most common error was “ `'...' was not declared in this scope` ” which means that a variable or function has been used that does not exist in this scope. This is not an error that would have been prevented if students were programming in Python. Students who do not understand scopes will most likely not have an easier time in Python. But some other common compiler errors

were related to missing brackets, braces, parenthesis and semicolons. These are the kind of errors that are less of an issue in Python. This can also be supported from the literature and specifically Koulouri et al. [16] who quantitively measured the amount of bugs students made (due to syntax) when comparing Java with Python. Students using Python will make less syntactical errors.

For warnings we see that the most common warning was `Wsign-compare`. This a warning that is shown when a signed value is compared with an unsigned value. The assignments during the course do not give any problems with this. The values that the students are supposed to use will not be big enough to cause problems when comparing signed and unsigned values. So which other warnings would actually occur less in Python? Some warnings such as the switch ones or missing return type would not appear in Python as Python has no switches and you do not need to return anything in a function. However, this can also raise potential Python specific issues. If we look at the following example.

No return statement on line 3:

```
1 def change(number):
2     value = number + 1
3     # No return statement on this line
4
5 x = 5
6 x = change(x)
7 print(x) # prints None
```

If students forget the return statement, Python would return "None" by default. So even though Python does not see a problem with a function that does not have a return value, it might not have been the intention of the programmer.

The only two warnings from our most commonly made warnings list that are relevant when comparing to Python is `Wparentheses` and `Wunused-variable`. The first is a warning when you omit parenthesis and the code might be confusing. For example, the GCC manual uses this example as confusing code [11].

Wparentheses warning example from the GCC manual:

```
1 {
2     if (a)
3       if (b)
4         foo ();
5     else
6         bar ();
7 }
```

In this example it is not clear to which if statement the else belongs. It belongs to `if(b)`, but most programmers would think it belongs to `if(a)` due to the indentation. In Python, thanks to indentation, it would be clear to which if statement the else belongs.
The second warning is `Wunused-variable`. This is a warning when a variable is declared but never used again. An example of this is shown below.

Two Wunused-variable warnings in C++:

```
1 int x; // gives an error in Python
2 int y = 1;
```

Assuming that `x` and `y` are never used anymore, we would get two `Wunused-variable` warnings. One of them coming from variable `x` that has no value assigned to it. In Python a variable without a value would result in an error thus making it easier to detect and solve.

6.3 Cppcheck issues

Cppcheck found 434 errors in 420 files. As we already said, we can disregard the most commonly made error which was uninitialized struct members. So the second most made error was forgetting to initialize variables. This is an error that will lead to a run time error in Python while in C++ the code can still compile. Two other errors were `stlOutOfBounds` and `arrayIndexOutOfBounds` which means that a data structure has been accessed out of bounds. This could happen in Python, but a clear error is given and needs to be fixed before continuing. Once again C++ does not give an explicit error when accessing out of bounds and would still continue compiling.

When it comes to warnings there is not much to discuss as Python would not change anything as most warnings are due to semantical mistakes that could also occur in Python.

Cppcheck also found 1208 style issues. From the most commonly made style issues we see that `unusedVariable` is something that would occur less in Python. This style issue is shown when you create a variable without assigning it a value and do not use it. As discussed before Python would give an error that needs to be fixed while C++ would only give a warning.

6.4 Bettercodehub guidelines

We looked at three guidelines from Bettercodehub. These guidelines advised to write at most 15 lines of code in a function, keep the amount of branch

points to at most 4 and to use at most 2 parameters for a function. We see that assignments that were not done well grade wise also had a lot of issues in Bettercodehub. For instance, week 7 and week 13 were considered to be hard assignments by the teaching assistants and it also reflects back in Bettercodehub as much code of the students is violating the guidelines. In general we see that in weeks that students were not given template code to work further upon, the Bettercodehub results were also bad. So what would our gain be with Bettercodehub if the students were using Python? A clear gain would be with the guideline "Write short units of code". The syntax of Python would allow students to write shorter code in general. However, it is not clear if the other guidelines would directly improve or worsen under Python. From the literature or from our own data we cannot conclude anything.

6.5 Interview with the teaching assistants and questionnaire

It is perhaps unsurprising to see that the teaching assistants ranked the assignments in such an order that is mostly consistent with the grades. They are the ones that hand out the grades in the first place and also provide formative feedback. Roughly summarized, some issues that the students were struggling with according to the teaching assistants that can be less of an issue in Python are:

1. C/C++ style strings. The difference between these is not always clear for students. In Python there is only one type of string and thus eliminating the need to know both types of string in C++. The reason why students are taught both type of strings is that C style strings are a good example of how to work with arrays. In Python arrays can be replaced by lists and from the literature we can see that not many lecturers have had any issues with this.

2. Off by one errors. In Python a clear error is given if students tried to access something that is out of bounds. This is not the case in C++ and only a warning can be turned on with a compiler flag and students generally do not touch compiler flags unless instructed. Even then it is only a warning. Thus it has to debugged by themselves or by a teaching assistant that helps them.

3. Vectors. Some students did not understand why vectors were needed. Python does not have arrays built in or vectors and uses lists which can serve as an adequate replacement for most use cases. If lists are introduced and used from the start, students only need to know this one data structure and can achieve the same goals without learning new

syntax. From the questionnaire we do see that the teaching assistants indicated that the students have a good understanding of vectors. So it seems that the students eventually understand how to work with vectors.

We also see that for the first half of the course the teaching assistants ranked assignment 7, 6 and 4 as the hardest three assignments. Students had array out of bounds issues, C style strings problems and some general C++ syntax issues in these three assignments. These issues would be less prevalent in Python.

When it comes to the second half of the course the three hardest assignments were 13, 12 and 9. However, here we see that the teaching assistants noticed less syntax issues and mostly problem solving struggles among the students. This can be explained by the fact that during these weeks no new syntactical constructs were introduced and students were not introduced to new data structures as well.

When we look at the questionnaire that the teaching assistants filled in after the interview, a few weak points of students seemed to be:

1. Students struggle with console input/output.
2. Students struggle with call by value/reference.
3. Students do not understand enumerations well.
4. Students do not understand how to work with files well.
5. Students struggle with C/C++ style strings.
6. Students do not write good structured code. (Good structured code is somewhat subjective, but this was supposed to a subjective opinion from a teaching assistant's perspective)

From point 1 we see that students struggle with console input/output. In Python the syntax of console input/output is shorter and can potentially help students understand it better. There is also no << or >> which students might not understand in the beginning.

Console input in Python:

```
1 x = input("Enter a number")
2 print(x)
```

Console input in C++:

```
1 int x;
2 cout << "Enter a number";
3 cin >> x;
4 cout << x;
```

Point 2 is a bit different as in Python arguments are passed by assignment. In Python students would have to understand the way Python passes arguments and the difference between a mutable and immutable object. Whether

this would be easier or harder than learning the difference between call by value/reference for our students is up for speculation. From the literature only Oldham [21] has mentioned this model of passing arguments to be a disadvantage as teaching the traditional distinction between call by value/reference is not possible anymore and had to be addressed in a follow up course.
Point 4 is related to working with ifstream/ofstream objects and files. The teaching assistants also mentioned this during the interview. Students had issues with ifstream/ofstream objects and also issues with reading in the contents of text files into data structures. As with point 1, once again Python provides shorter code for this which can help students understand it easier.
Point 5 has already been discussed above.
The last point is also an interesting one. As we have seen in the literature review, most researchers talked about the simple syntax of Python that also helped students to write better code. In our course the teaching assistants are the ones that have to help students during the lab sessions and grade their assignments. Spending too much time reading unreadable and unclear code will only slow down these teaching assistants. Thus resulting in them being unable to help other students or giving less useful feedback. It is also not uncommon to see that students have a varied style of writing their brackets or other coding styles. Python enforces the coding style a lot more. Which would not only help students to write more readable code, but also help the teaching assistants in their ability to help the students.

On the other hand, from the questionnaire we see that the teaching assistants did not notice students struggling with the following.

1. Students understand functions, both void functions and functions with a type.
2. Students do not struggle with types and operators. Although students did have issues during with the assignment where they had to overload operators according to the interview.
3. Students understand control structures.
4. Students have a good understanding of scope, both local and global. Although we did see that the most common compiler error was due to scope issues.
5. Students understand data structures such as arrays, vectors and structs. From the interview we did hear that students struggle during the assignment with these data structures, but from the questionnaire the teaching assistants seemed to agree that students eventually understood them.

So according to the teaching assistants, many programming concepts are

understood well by students in C++. As we saw in the literature review, there were some researchers that considered the dynamic typing of Python to be a disadvantage. So we could run into some Python typing issues due to dynamic typing if the students were programming in Python. Dynamic typing is more flexible, but it also hides information. Thus potentially creating other problems.
Another point is that most data structures in C++ are eventually understood well by students. Many researchers have mentioned that the data structures of Python were useful in their classes. This along with the simple syntax of Python made it easier for students to use them. According to the teaching assistants our students do struggle with data structures in C++ such as making off by one errors, but eventually data structures are understood well by the students. As discussed before, Python would be able to help students spot off by one errors a lot better.
Control structures were also well understood by students. From the literature we saw that Koulouri et al. [16] and Jayal et al. [15] mentioned that their students used more keywords (if, while, for) in their assignments. However, our students do not seem to struggle with these control structures.

The last two remaining statements that we have not discussed yet is "students write readable code" and "students spend more time problem solving than understanding syntax". From the answers of the teaching assistants there is no clear agreement or disagreement. For the first statement this can be explained from the fact that readable code is subjective. For the second statement it can be that the teaching assistants help many students with various problems which makes it hard for them to judge whether problem solving or syntax is their real issue. Also some common syntax issues such as missing semicolons or brackets are easy to fix for students themselves without the intervention of a teaching assistant.

Chapter 7

Related Work

In section 3 we extensively discussed many papers that focused on Python. Here we will go over some other relevant papers that focus less on Python.

Almost 40 years ago in 1980, Wexelblat [38] researched some effects of one's first programming language. His hypothesis was that your first programming language somehow affected your future programming ability. Thus he went on to ask in various journals for people to share their own experiences and answer a few questions of his. Questions such as whether they considered x language (to be filled in by the responder) was a good first programming language or how a first programming language might impact future programming ability. Although Wexelblat himself says that firm conclusions cannot be drawn as this was not a formal research we can still see some interesting results. Most responders did agree with him that your first programming language affects you in the future. They also considered that a "small" language would be good as a first programming language. Wexelblat's research was not in depth or detailed, but we can see that even 40 years ago some programmers already saw the effects of your first programming language.

In 2008 Vujošević-Janičić and Tošić [36] looked at programming paradigms in CS1. They discussed various points when it comes to choosing a paradigm and language for CS1. It is interesting to see that at that point Python was not as popular as it is right now. The authors only mentioned Python in a few sentences and also indirectly with the sentence: "In any case, script-languages are becoming significant and soon they will have an important role in teaching programming". They were not wrong. As pointed out by Guo [12], in 2014 Python became the most popular CS1 language at top American universities.

Farooq et al. [10] in 2014 created a framework to compare popular first programming languages. This framework works as follows. Every language

is scored on 13 features. This consists of seven technical features that are related to a language and six external features such as the popularity of a language in the industry. Every language receives a final numerical score and this is how "good" a first programming language is. This framework also allows a user to assign weights to certain features themselves and thus be able to put importance on the features that they want in a first programming language. With the default weights the languages Java and Python come out on top.

There are also papers [1, 14] that researched commonly made compiler errors by novice students. Jadud [14] studied the compilation behavior of his students in an attempt to find commonly made errors. These students took an introductory object-oriented Java course. Jadud collected compiler errors from the students during the weekly lab sessions. Some of the commonly made errors were missing semicolons, using unknown variables and missing brackets. Even though the students of Jadud used a different language, paradigm and the data was collected differently compared to us, the errors Jadud found are somewhat similar to the most commonly found compiler errors of our students. These were small syntactical errors such as missing certain characters like brackets or semicolons.

Altadmri and Brown [1] did something similar, but with a larger sample size. They had access to roughly 37 million compilations from over 250.000 students that programmed in Java. This large sample size and analysis gives a reasonable idea of commonly made compiler errors by students in Java. Once again a direct comparison with our found results is not possible, but it is interesting to see that the third most compiler error they found was "Control flow can reach end of non-void method without returning." which means that somewhere a return statement is missing in a function. This is similar to the third most compiler warning `Wreturn-type` that we found with our students. The authors also looked at how long it took to fix certain compiler errors. They conclude that it was mostly semantical errors that took longer to fix. This is something that Denny et al. [7] have also looked at.

Chapter 8

Conclusions

In this thesis we explored Python as a potential replacement for C++ in the course Imperative Programming for Computing Science at Radboud University. We first performed a literature review and extracted advantages/disadvantages of using Python in an introductory programming course. We found that many researchers considered the syntax of Python to be the most important advantage. Some other features of Python such as the data structures (especially lists), interpreter and dynamic typing were also considered to be advantageous. When it comes to disadvantages, some authors mentioned dynamic typing, the lack of call by value/reference and the absence of arrays. We also analyzed the assignments of students in the course Imperative Programming during study year 2018/2019. 1783 assignments were analyzed for compiler errors, compiler warnings, Cppcheck issues and some Bettercodehub guidelines. Following this we interviewed the teaching assistants to give us a more complete image of the struggles of students.

8.1 Final recommendation

Our main research question was:

- **Could Python potentially be a more suitable first programming language for Computing Science at Radboud University?**

From the research in this thesis we conclude that Python could be more suitable for the following reasons.

1. Common compiler errors made by students that hand in failing assignments are related to syntax issues that would occur less in Python.
2. Students eventually obtain a good understanding of most data structures in C++, but during this learning process a common problem they

encounter is off by one errors when accessing arrays/vectors. Python can help students detect these errors better than C++.

3. Some common Cppcheck issues made by students such as uninitialized variables, unused variables would also be easier to detect in Python.

4. In general, the simple syntax of Python can ensure that students spend more time on problem solving instead of understand the syntax of a language. For example, students struggled with console input/output. The syntax of Python with console input/output is shorter in comparison with C++ and thus this can help students to understand it faster.

However, some care should be taken with the way that Python passes arguments and dynamic typing. These can potentially bring up issues that need to be addressed and explained to students.

8.2 Future work

We end our conclusion with some potential future work.

1. A continuation of this thesis can be considered. The course Imperative Programming for computing science students at Radboud University is given during the first semester of a year. As this thesis was conducted during the second semester it was not possible anymore to collect data directly from the students. Relevant data that could be collected is the compilation data of the students when they are working on the assignments. Some researchers have done similar things [14, 1] and can serve as a reference point. Furthermore, qualitative data from the students could be collected with interviews or questionnaires. It would especially be useful to collect data from students that failed the course or dropped out.

2. A similar Imperative Programming course for the artificial intelligence students is given in Java instead of C++. If possible in the future, collecting data from those students and comparing it with the students that take Imperative Programming at Computing Science could give us a useful comparison.

3. When considering a change in programming language for the course Imperative Programming, collect relevant data in advance for future comparison. Or even better, split the course in two groups when the change in programming language happens. One group would be taught using the new language and the other group C++. This would provide good comparison material.

4. Following on the last point, similarly to Enbody [8, 9], measuring results in future courses from two groups of students who used a different first programming language would be interesting as well. Computing science students generally take the course Object Oriented Programming in Java during their second semester along with a course Hacking in C in which C/C++ knowledge is expected. Those two courses could potentially be impacted the most.

Bibliography

[1] Altadmri, A., and Brown, N. C. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2015), SIGCSE '15, ACM, pp. 522–527.

[2] Ateeq, M., Habib, H., Umer, A., and Rehman, M. U. C++ or Python? Which One to Begin with: A Learner's Perspective. In *2014 International Conference on Teaching and Learning in Computing and Engineering* (apr 2014), IEEE.

[3] Cern. Cling. https://root.cern.ch/cling.

[4] Codeblocks. http://www.codeblocks.org/downloads/26.

[5] Cppcheck. http://cppcheck.sourceforge.net/.

[6] Cppcheck. Cppcheck manual. http://cppcheck.sourceforge.net/manual.pdf.

[7] Denny, P., Luxton-Reilly, A., and Tempero, E. All syntax errors are not equal. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12* (2012), ACM Press.

[8] Enbody, R. J., and Punch, W. F. Performance of python CS1 students in mid-level non-python CS courses. In *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10* (2010), ACM Press.

[9] Enbody, R. J., Punch, W. F., and McCullen, M. Python CS1 as preparation for C++ CS2. *ACM SIGCSE Bulletin 41*, 1 (mar 2009), 116.

[10] Farooq, M. S., Khan, S. A., Ahmad, F., Islam, S., and Abid, A. An Evaluation Framework and Comparative Analysis of the Widely

Used First Programming Languages. *PLoS ONE 9*, 2 (feb 2014), e88941.

[11] GNU. Warning options. https://gcc.gnu.org/onlinedocs/gcc-5.1.0/gcc/Warning-Options.html#Warning-Options.

[12] Guo, P. Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. *BLOG@CACM* (2014).

[13] Hunt, J. M. Python in CS1 - Not. *J. Comput. Sci. Coll. 31*, 2 (Dec. 2015), 172–179.

[14] Jadud, M. C. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education 15*, 1 (mar 2005), 25–40.

[15] Jayal, A., Lauria, S., Tucker, A., and Swift, S. Python for Teaching Introductory Programming: A Quantitative Evaluation. *Innovation in Teaching and Learning in Information and Computer Sciences 10*, 1 (feb 2011), 86–90.

[16] Koulouri, T., Lauria, S., and Macredie, R. D. Teaching Introductory Programming. *ACM Transactions on Computing Education 14*, 4 (dec 2014), 1–28.

[17] Leping, V., Lepp, M., Niitsoo, M., Tõnisson, E., Vene, V., and Villems, A. Python Prevails. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing* (New York, NY, USA, 2009), CompSysTech '09, ACM, pp. 87:1–87:5.

[18] Luxton-Reilly, A., Sheard, J., Szabo, C., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., and Scott, M. J. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018 Companion* (2018), ACM Press.

[19] Miller, B. N., and Ranum, D. L. Teaching an introductory computer science sequence with Python. In *Proceedings of the 38th Midwest Instructional and Computing Symposium, Eau Claire, Wisconsin, USA* (2005).

[20] Murphy, E., Crick, T., and Davenport, J. H. An Analysis of Introductory Programming Courses at UK Universities. *The Art, Science, and Engineering of Programming 1*, 2 (apr 2017).

[21] Oldham, J. D. What Happens After Python in CS1? *J. Comput. Sci. Coll. 20*, 6 (June 2005), 7–13.

[22] Patterson-McNeill, H. Experience: From C++ to Python in 3 Easy Steps. *J. Comput. Sci. Coll. 22*, 2 (Dec. 2006), 92–96.

[23] Python.org. How do I write a function with output parameters (call by reference)? https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference.

[24] Python.org. Python documentation. https://docs.python.org/3/reference/datamodel.html.

[25] Python.org. Using the Python Interpreter. https://docs.python.org/3/tutorial/interpreter.html#the-interpreter-and-its-environment.

[26] Python.org. What is Python. https://docs.python.org/3/faq/general.html#what-is-python.

[27] Shannon, C. Another Breadth-first Approach to CS I Using Python. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2003), SIGCSE '03, ACM, pp. 248–251.

[28] Stroustrup, B. FAQ. http://www.stroustrup.com/bs_faq.html#invention.

[29] Stroustrup, B. FAQ. http://www.stroustrup.com/bs_faq.html#why.

[30] Stroustrup, B. Bjarne Stroustrup: Why I Created C++. https://www.youtube.com/watch?v=JBjjnqG0BP8, June 2011.

[31] TIOBE. Tiobe index. https://www.tiobe.com/tiobe-index/.

[32] van Rossum, G. Why was Python created in the first place? https://docs.python.org/3.7/faq/general.html#why-was-python-created-in-the-first-place.

[33] van Rossum, G. Foreword for "Programming Python". https://www.python.org/doc/essays/foreword/, May 1996.

[34] van Rossum, G. Polderpioniers: Guido van Rossum, de man achter Python. https://www.youtube.com/watch?v=USTL2gxhRkg&t=, December 2015.

[35] Visser, J., Rigal, S., Wijnholds, G., Eck, P. v., and van der Leek, R. *Building Maintainable Software, C# Edition: Ten Guidelines for Future-Proof Code*, 1st ed. O'Reilly Media, Inc., 2016.

[36] Vujošević-Janičić, M., and Tošić, D. The role of programming paradigms in the first programming courses. *The Teaching of Mathematics*, 21 (2008), 63–83.

[37] Wainer, J., and Xavier, E. C. A Controlled Experiment on Python vs C for an Introductory Programming Course. *ACM Transactions on Computing Education 18*, 3 (aug 2018), 1–16.

[38] Wexelblat, R. L. The consequences of one's first programming language. *Software: Practice and Experience 11*, 7 (jul 1981), 733–740.

[39] Zelle, J. M. Python as a First Language. *Department of Mathematics, Computer Science, and Physics Wartburg College* (1999).

Appendix A

Appendix

A.1

Some code is provided as example of the compiler errors seen in figure 5.3. The code is found on Github https://github.com/huynguy97/Thesis-data/blob/master/compileErrors.cpp.
An explanation of the errors can also be found below.

1. `'...' was not declared in this scope` = Using a variable or function that does not exists in this scope.
2. `expected '...' before '...'` = Common error such as missing a semicolon before the next line of code.
3. `expected '...' before '...' token` = Similarly to above, but this time a character is expected before another character. Such as missing a semicolon before a closing } bracket.
4. `no match for '...' (operand types are '...' and '...' )` = This error is when an operator has been used that is not defined yet for certain types.
5. `expected primary-expression before '...'` = The compiler expected a literal.
6. `expected primary-expression before '...' token` = Similarly to above, but this time a primary-expression was expected before a certain character.
7. `expected unqualified-id before '...'` = For example:

unqualified-id error:

```
1 void test();{
2     cout << "There should be no ; before {";
```

```
3 }
```

8. `expected '...' at end of input` = Common error when missing a closing character such as }.

The rest compiler errors from figure 5.3 can be found in the following picture.


```
error: cannot convert to in return': 10, ' error: no matching function for call to ': 10, ' error: invalid initialization of
reference of type from expression of type ': 10, ' error: has no member named ': 9, ' error: could not convert from to ': 9, '
error: too few arguments to function ': 8, ' error: a function-definition is not allowed here before token': 7, ' error: expected
declaration before token': 7, ' error: does not name a type': 6, ' error: expected initializer before ': 6, ' error: cannot convert
to for argument to ': 6, ' error: return-statement with a value, in function returning [-fpermissive]': 5, ' error: expected
unqualified-id before token': 5, ' error: expected or before token': 5, ' error: template argument 1 is invalid': 4, ' error:
template argument 2 is invalid': 4, ' error: invalid initialization of non-const reference of type from an rvalue of type ': 3, '
error: expected identifier before token': 3, ' error: invalid conversion from to [-fpermissive]': 3, ' error: expression list
treated as compound expression in initializer [-fpermissive]': 3, ' error: request for member in , which is of non-class type ': 3, '
error: conversion from to non-scalar type requested': 3, ' error: redefinition of ': 2, ' error: cannot resolve overloaded function
based on conversion to type ': 2, ' error: cannot convert to in assignment': 2, ' error: switch quantity not an integer': 2, ' error:
cannot convert to in initialization': 2, ' error: without a previous ': 2, ' error: initializer provided for function': 2, ' error:
  declared as reference but not initialized': 2, ' error: declaration of shadows a parameter': 2, ' error: too many arguments to
function ': 2, ' error: variable or field declared void': 2, ' error: pch.h: No such file or directory': 1, ' error: has not been
declared': 1, ' error: resource.h: No such file or directory': 1, ' error: cannot bind lvalue to ': 1, ' error: missing terminating "
character': 1, ' error: cursor.h: No such file or directory': 1, ' error: scalar object requires one element in initializer': 1, '
error: expected initializer before token': 1, ' error: conversion from pointer type to arithmetic type in a constant-expression': 1,
' error: expected primary-expression at end of input': 1, ' error: invalid operands of types and to binary ': 1, ' error: expected
statement at end of input': 1, ' error: stray in program': 1, ' error: invalid cast from type to type ': 1, ' error: cannot be used
as a function': 1, ' error: forming reference to void': 1, ' error: invalid use of ': 1, ' error: expected after struct definition': 1,
' error: expected or before ': 1, ' error: unable to find numeric literal operator ': 1, ' error: invalid use of member (did you
forget the ?)': 1, ' error: conflicting declaration ': 1, ' error: unterminated argument list invoking macro "assert"': 1, ' error:
is not a type': 1, ' error: invalid types for array subscript': 1, " error: no match for (operand types are and
'__gnu_cxx::__alloc_traits<std::allocator<std::__cxx11::basic_string<char> > ": 1, ' error: macro "assert" passed 2 arguments, but takes
just 1': 1, ' error: called in a constant expression': 1, ' error: body of constexpr function not a return-statement': 1, ' error: ISO
C++ forbids comparison between pointer and integer [-fpermissive]': 1, ' error: use of enum without previous declaration': 1, ' error:
the value of is not usable in a constant expression': 1, ' error: redeclaration of ': 1, ' error: no match for call to ': 1
```


A.2

We also refer to the GCC manual which provides clear explanations of the warnings. https://gcc.gnu.org/onlinedocs/gcc-5.1.0/gcc/Warning-Options.html#Warning-Options
The options that we used to perform our analysis are:

1. `Wall` = This option does not turn on all warnings, but it turns on a lot of warnings that GCC considers to be useful and easy to avoid.
2. `Wextra` = Turns on some extra warnings.
3. `Wswitch-default` = Warns when missing a default case in a switch.
4. `Wswitch-enum` = Warns if all possible options of an enumeration are not in the switch.

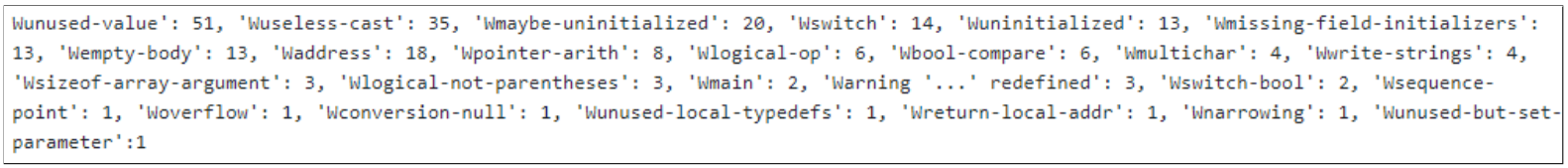
5. `Wshadow` = Warns for shadowing. For example, when a local variable has been declared that has the same name as a global variable and might cause confusion.
6. `Wlogical-op` = Warns for possible incorrect or suspicious use of logical operators.
7. `Wuseless-cast` = Warns when something is casted to the same type.

A.3

Some warnings have already been explained above. Some other warnings that we see on the figure will be explained here.

1. `Wsign-compare` = Comparing a signed and unsigned value could lead to incorrect results.
2. `Wreturn-type` = Missing a return statement in a non-void function or when a return type defaults to int.
3. `Wunused-variable` = A variable is declared but never used.
4. `Wunused-parameter` = A function parameter is never used.
5. `Wtype-limits` = Warns when a comparison is always true or false.
6. `Wunused-but-set-variable` = Similar as the `Wunused-variable` warning, but this time the variable is assigned a value.
7. `Wparentheses` = Warns for potential confusing use of parenthesis. See also the example in the discussion here.

The rest compiler warnings from figure 5.4 can be found in the following picture.



A.4

When it comes to Cppcheck some errors are rather self explanatory or have a good explanation on the site https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/. In the following sections we will try to clear up some messages in case some of them are confusing. Everything between quotes is from Cppcheck and anything else is our own additional explanation.

1. `uninitStructMember` = When not initializing all struct members this error shows up.
2. `uninitvar` = "Uninitialized variable."
3. `stlOutOfBounds` = For example, when trying to access an index out of bounds of a string this error will show up. The difference between this and something such as array index out of bounds is that these are errors when using the standard template library (STL) incorrectly.
4. `syntaxError` = Various syntax errors. An example being not declaring the type of a function.
5. `invalidFunctionArgBool` = Passing a boolean argument to a function that requires something else, such as the abs example in the section 2.2.
6. `arrayIndexOutOfBounds` = Accessing arrays out of bounds.

The rest Cppcheck errors from figure 5.5 can be found in the following picture.

```
'negativeContainerIndex': 1, 'coutCerrMisusage': 1, 'returnReference': 1, 'internalAstError': 1, 'containerOutOfBounds': 1
```

A.5

1. `identicalConditionAfterEarlyExit` = As seen in the following example, the first if statement is identical to the return statement.

identicalConditionAfterEarlyExit:

```
1 if ( year%400 == 0 )
2     return true ;
3 if ( year%100 == 0 )
4     return false ;
5 return year%400 == 0 ;
```

2. `compareBoolExpressionWithInt` = "Comparison of a boolean expression with an integer."
3. `incorrectLogicOperator` = For example, when a "logical disjunction always evaluates to true" Cppcheck will give this warning. Incorrect usage of boolean operators that lead to the same boolean values on both sides are the main cause of this warning.
4. `identicalInnerCondition` = This happens when both conditions are the same when entering something such as the while loop on the next page. In the example we see that both the conditions of the while and if statement are the same.

identicalInnerCondition:

```
1  while(r>0)
2  {
3      if(r>0)
4      {
5      ...
```

5. `selfAssignment` = "Redundant assignment of '...' to itself."
6. `incorrectCharBooleanError` = "Conversion of char literal to bool always evaluates to true."
7. `oppositeInnerCondition` = "Opposite inner condition leads to a dead code block." This is the opposite of identicalInnerCondition.
8. `sizeofwithsilentarraypointer` = "Using '...' on array given as function argument returns size of a pointer."

The rest Cppcheck warnings from figure 5.6 can be found in the following picture.

```
'negativeContainerIndex': 3, 'redundantAssignInSwitch': 1, 'argumentSize': 1, 'staticStringCompare': 1, 'assertWithSideEffect': 1,
'ignoredReturnValue': 1, 'sizeofCalculation': 1, 'assignmentInAssert': 1
```

A.6

1. `catchExceptionByValue` = "Exception should be caught by reference." Remember that this style issue was already in the template code of week 1 and 2 and can be ignored.
2. `unreadVariable` = "A variable is assigned a value that is never used." This should not be confused with `unusedVariable`.
3. `shadowFunction` = "Local variable shadows outer function."
4. `variableScope` = "The scope of the variable can be reduced."
5. `unusedVariable` = "Unused variable." So there is also no assignment to this variable. This should not be confused with `unreadVariable`.
6. `duplicateAssignExpression` =

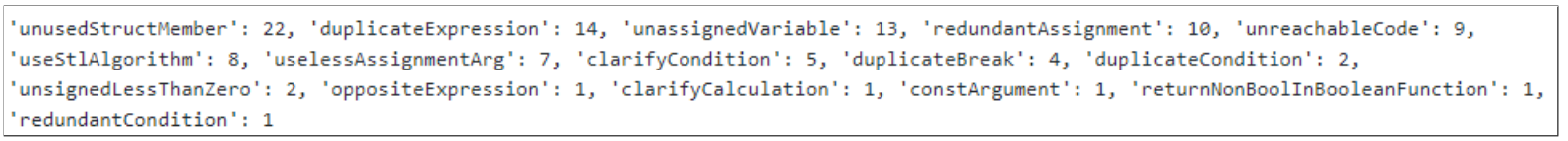
duplicateAssignExpression

```
1  blue = last+1;
2  int white = last+1;
```

This is a style issue due to two of the same assignments happening after each other. It can be correct, but Cppcheck warns the programmer to check this just to be sure.

7. `shadowVar` = "Local variable shadows outer variable."
8. `knownConditionTrueFalse` = "Condition is always true" or "Condition is always false"
9. `arrayIndexThenCheck` = "Array index is used before limits check."
10. `unsignedPositive` = "Unsigned expression can't be negative so it is unnecessary to test it."

The rest Cppcheck style issues from figure 5.7 can be found in the following picture.



A.7

Bettercodehub has ten guidelines. We only consider three guidelines to be relevant. The other seven guidelines are:

1. `Write code once` = This guideline warns when the same blocks of code is found in multiple places. This indicates that a function could have been created instead of repeating the same blocks of code. Although this was a relevant guideline, for practical reasons it could not be included. Bettercodehub was used in such a way that analyzing 1783 individual assignments would be impractical.
2. `Separate concerns in modules` = This guideline warns when a certain class is being called too much and is not relevant as students do not use classes.
3. `Couple architect components loosely` = This guideline warns when there is too much interface code and once again not relevant as students do not use classes or interfaces.
4. `Keep architecture components balanced` = This guideline warns when the size of classes differ too much relatively. This is not relevant as students do not use classes.
5. `Keep your codebase small` = Although this was somewhat relevant, the "codebase" of students consisted of a single assignment that could be finished in hours. This guideline is more relevant for big projects and not smaller weekly assignments.
6. `Automate tests` = Not relevant as students do not write any tests.

7. `Write clean code` = This guideline warns for useless comments, commented code blocks and dead code. As comments have been removed for anonymization reasons this guideline is mostly not relevant. Dead code is relevant, but the majority of the guideline was not and thus it was not considered.