UNIVERSITETET
I OSLO

Masteroppgave

How to identify and correct misconceptions.

Students' understanding of references in Python.

Boye Mathias Molteberg

Informatics: Programming and System Architecture
60 ECTS

Department of informatics
Faculty of mathematics and natural science

Abstract

References are often perceived as a difficult topic both to teach and to learn. However, they are one of the most important topics to master. References or rather reference variables are imperative to a well-functioning object-oriented program. Allowing for the mutation of a single object to be accessed from all variables that retain a reference to the object.

This study explores what motivates a better understanding of references among novices in Python. The novices that are involved in the study are students in the course IN1000 at the Department of Informatics at the University of Oslo. As students understanding of references in the course already have been explored, it is of interest to see if and when the students' mental models merge into one singular correct mental model.

This thesis aims to contribute to the improvement of science education by unveiling when the correct mental model in conceived by the participants of the study. Following two novices, Sam, and Alex, learning programming for the first time. Following their progression and how their understanding of variables and references change over time. How Sam and Alex learn references can provide insights into how novices learn references and what causes them to learn. These insights gained by following and tutoring Sam and Alex are summarized as three key points needed to teach references effectively. These three key points can help teachers and students to better teach and learn references.

Acknowledgments

Firstly, I would like to thank my thesis advisors, Associate Professor Ragnhild Kobro Runde and Lecturer Siri Annethe Moe Jensen, for their help and corrections. Without their help, encouragement and clarifying conversations, it would have been a lot harder to write this thesis and transfer my thoughts to paper.

I would also like to thank the participants in the observations, group teachings and especially the two students that allowed me to follow their learning. Without the engagement and openness this thesis would not be possible.

Lastly, I would like to thank my friends and family for their understanding and encouragement.

Author

Boye Mathias Molteberg

Contents

Table of Tables

Table of Figures

1. Introduction

Modern, object-oriented programming languages, rely on objects to handle values and conduct operations. Learning how to use and make objects can be a daunting for a novice that have just recently started learning programming. Learning to use objects requires the knowledge of how references and reference variables, often just called references and variables, act.

What are references and why do they matter in programming? A reference points to an object in the memory, therefore often called reference variables, aliases or pointers. As the assignment and use of references can get quite complex, understanding them can come with misconceptions. It is therefore useful to identify how to approach these misconceptions. This thesis will look at how to better identify and correct these misconceptions.

A misconception can be caused by one or more inaccurate mental models. A model approximates the real world, and as such a mental model is the model used by an individual to conceptualize a concept. A correct mental model will always give a correct result, but it does not mean that a wrong model always gives the wrong answer. The mental models that the programmer has, can change based on new knowledge. It can be difficult for a programmer, novice, or otherwise to detect if these misconceptions apply to them. Some students struggle to apply the correct mental model to references in Python[22], and these same students will likely struggle with the activity of tracing new programs. Based on studies from the Leeds working group, there is a correlation between students that can explain code well and can produce meaningful and accurate code[16, 25]. This explanation is done mainly by tracing the program. As understanding references is a component of tracing code, since the changes to an object can be accessed through multiple reference variables, not understanding reference can affect the ability to trace code. We can assume that students that don't understand references will struggle with tracing. Therefore, it is reasonable to believe that; Programmers that do not understand references cannot understand code that uses objects.

Based on the study of mental models [22], the study on swapping variables [23], and two articles on tracing [16, 25], the assumption seems to be that a lack of understanding of references coincides with a poorer understanding of coding in general. As references are a large part of understanding how objects behave in Python, understanding references can cause a poorer understanding of code in general.

In order to aid students with understating references, it is not only important to understand what mental models occur, or what the consequences are, but also when and why they occur. By understanding this, we can get closer to revealing why the wrong mental models occur.

1.1. Research question

The goal of this thesis is to discover how students develop their understanding of references over time, and what influences this change. Given the importance of objects and the manipulation of them in every object-oriented language, the understanding of references and by extension object is critical in writing code that run and is easy to maintain. References are crucial to properly handle objects. The concept can be difficult to fully grasp, both by novices and more experienced programmers. Therefore, it is useful to know how novices to programming learn and develop their understanding of references.

1. How do novice programmers understand references and variables?
    a. How do novices understand how the code acts?
    b. How do novices perceive objects and references?
2. How does the understanding of references change over time?
    a. What causes the change?
    b. Is the change triggered by lecturing and tutoring, or is it derived from the novice's own experiences?
    c. Are the novices aware of the change in their own understanding?
3. What causes positive change toward a more complete mental model of references?
    a. What actions can novices take to better their own understanding?
    b. What actions can educators take to assist students with understanding references?

1.2. Research setting

The participants of the study were all enrolled in the IN1000 course. IN1000 teaches Python to new students at the Department of Informatics at the University of Oslo. These new students are often new to the university, they often start directly after high school. While many students in the course are new, some of the students have previous higher education or work experience.

The course is an introductory course for students that will learn programming and is intended as a foundation for later programming courses. The next programming language most students will learn is Java trough IN1010, and therefore the focus of IN1000 is to teach skills that are transferable between programming languages. The interviews, observations and tutoring was dictated using the lecture program and was conducted as shown in Figure 1.

1.3. Structure

The rest of the thesis is structured as follows. Chapter 2 will present relevant background literature, frequently used terms and research on mental models, tracing, and notional machines. Chapter 3 will present the research methods used. The methods will explain the methodology and practical steps in observing, interviewing, and tutoring the participants. Chapter 4 going through the results and analysis from the interviews and tutoring with two students, Sam, and Alex. As well as the results and analysis of the observations and group teaching. The analysis will be more of a summary of the results and be utilized in the discussion at chapter 5 where the research questions will be discussed based on the experiences made through the observations, interviews, and tutoring. Chapter 6 will contain the conclusion from the thesis.

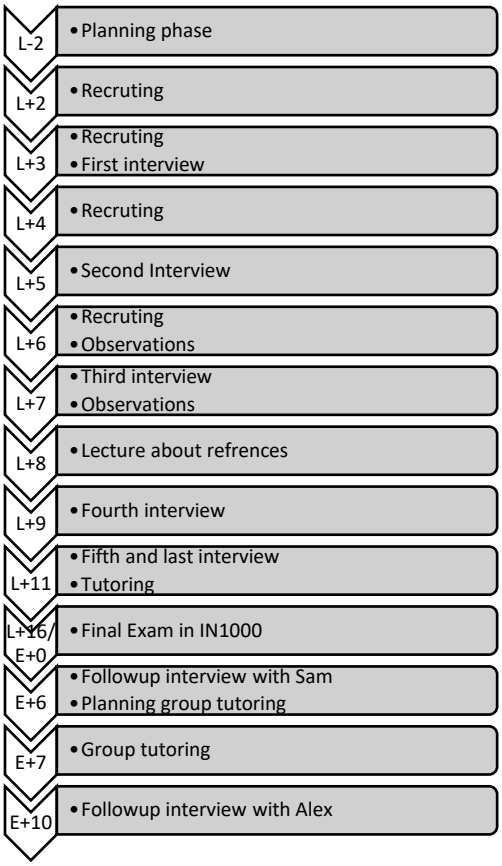


Figure 1: Study timeline.

2. Background

This chapter will focus on the relevant literature for researching mental models for references in Python. Fist a description of references and objects will be presented in subsection 2.1. The chapter will continue to introduce the mental models around the use of references in subsection 2.2. The subsection will be using primarily the article from Rørnes [22] as it had a good explanation of the different mental models. The next subsection will present the relevant literature for Notional Machines in 2.3 and presenting the relevant notional machines in the subsection 2.3.1. The use of the SOLO taxonomy will be presented in subsection 2.4 explaining each type of response in each the subsections of 2.5. In subsection 2.5 will present the literature found when discovering the work of Lister [14] using a literature search in Google Scholar. As the article introduced both the efforts of the Leeds working group with the BRACElet study[15, 16, 25, 26] as well as the taxonomies of SOLO[14, 26, 28] and BLOOM[12, 18, 26]. Learning from their experiences working with tracing tasks and code understanding. The literature search also found a study focusing on profiles and mental models over time by Henry and Dumas [5]. While Henry and Dumas does not explicitly follow the development of references over time. The results from Henry and Dumas show a development over time, and that students learn differently. The background will lastly present the Dunning-Kruger effect, a popular teaching theory, in section 2.6.

2.1. What are references and objects?

Although references and objects are used frequently by any object-oriented programmer, a lot of professionals disagree how to describe them, and their relation. A useful description of an object reference, often only called reference, is that they identify where an object is located. They contain the address of the object but not the object itself. An object being a container for values, sometimes only primitive values like strings, integers etc. other times they contain more complex values like interfaces, web pages and other objects. [6]

One of the issues with teaching references is the difference in terminology used. Some languages and educators use words like pointers, object containers and others to describe references, some don’t mention them at all. The issue is further complicated as to make a change on an object and on a variable is described the same way. This can cause confusion for a novice and not knowing the difference between an object and a variable.

In Simula, Java and Beta, a reference are stated explicitly, as opposed to Python's implicit statement. In Simula, assigning a reference to a variable is called a reference assignment, and use a different assignment statement :− as opposed to assigning a primitive value with the use of :=. When explaining references, both the word pointer and reference is used [11].They also separate references from reference variables as shown in Figure 2. [1, 11]

```
ref(Object) refrence_variable.
refrence_variable :- new Object;
```

Figure 2: Code example of reference assignment in Simula.

In Beta the difference is between a static reference and a dynamic reference. The static reference will be constant as it always denotes the same static object. A dynamic reference or also called a variable reference is declared as *A: ^T* while a static reference is declared as *X: @T*. The reference, *A* and *X*, contains a pattern. In the same way a reference can be assigned in the way *X[]→ A[]* or *120 → X*. In the first example the reference A now points to the same object as X does. The second example makes the reference *X* refer to the object *120*, a static integer.[13]

In Java, references are called both called pointers[7, 19] and references[20], depending on the author. An example of creating a reference to a non-static object is shown in Figure 3. As Beta and Simula, the difference between a primitive and an object is called differently. While the difference primarily is the use of new when calling the variable, it allows for static variables that are called directly.[7, 19, 20]

```
// References
Object variable = new Object()
// Non reference variable
Int variable = 1
```

Figure 3: Code example of reference variable with the object Object and the variable containing an Integer.

2.2. Mental models

What is a mental model? Mental models stem from the field of psychology and is a "small-scale model" of reality. It is this model that a practitioner in a field, in this thesis a programmer, use to conclude the results from a question [10]. In this thesis, the mental models are the models the novices use to explain how a piece of code runs in Python. Explained differently, how the novices approach a piece of code and arrive at a consistent result. There are many mental models that explain how a piece of code works in Python, to narrow the focus, the study will focus on the mental models held to explain references. What mental model do novices use to predict how a reference act and how reference variables behave.

In [23], Simon focuses on swapping variables, tested using tracing questions in the final exam. The results support the belief that variables are a critical part of tracing, as the knowledge of variables is a building block for understanding references. The findings give support to a hypothesis that some novice programmers do not understand and cannot correctly apply neither assignment statements, sequences of statements, or both. It seems reasonable to conclude that knowledge about assignment statements and sequences of statements is knowledge transferable to references. Simon goes through models on sequential variable swapping, with two models focusing on sequential and simultaneous variable swapping. Although these models are not directly transferable to the understanding of references, they indicate that mental models exist to explain how students perceive programming and how code executes.

2.2.1. Previous Research

The primary research used on Mental Models for references in Python in the article is produced by Rørnes et al[22] and Rørnes master thesis[21] on the topic. Their article arrived at two main categories, Copy Value Model (MVAL) and Copy Reference model(MREF). In this thesis the Copy Value Model and the Copy Reference Model will be shortened to CVM and CRM respectively for simplicity. There are two subcategories for Copy Value Model and Copy Reference Model, as the nuances of where the value is stored is different on the different sub categories. The subcategories will be numbered with the more correct mental model as nr. 1, as such the correct mental model will be CRM1 and copy value model closest to CRM1 will be CVM1.

```
1: a = Object(1)
2: b = a
3: a = Object(2)
```

Figure 4: Example code to explain the difference between the different mental models.

2.2.2. Copy Value Model (CVM)

The copy value model or CVM for short, is a mental model that says only the value of the variable is carried over. For some languages, and some instances, this is the correct way to think. For objects in Python, it will result in the wrong answer if the objects are mutated. There are two subcategories of CVM[21, 22] based on how the programmer thinks of value storage. It is not possible to differentiate in the answers produced by the different sub category, however they are conceptually different. There is not made a difference between

CVM1 and CVM2 in setting up the survey in the thesis. However, if a possibility to differentiate CVM1 and 2 is presented during the study, it will be utilized.

CVM is valid for immutable objects in Python and objects with a single variable pointing to them.

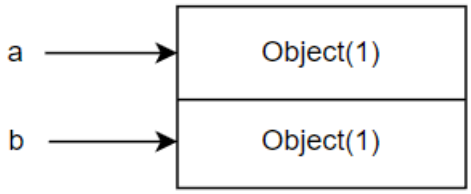


Figure 5: Line 2 of the example code using CVM. Based on figures from Rørnes et al.

Using the CVM model, with both sub categories, the code from Figure 4 should act as shown in Figure 5. Each variable having an identical, yet unique, copy of the object.

CVM1

In the first subcategory of CVM the value of the variable is reassigned to a new memory location[22], that a now refers to after line 3 is executed as shown in Figure 6. This is the more correct mental model of the CVM models. The understanding of where the object is stored is correct, however the concept of pointers is not known or considered.

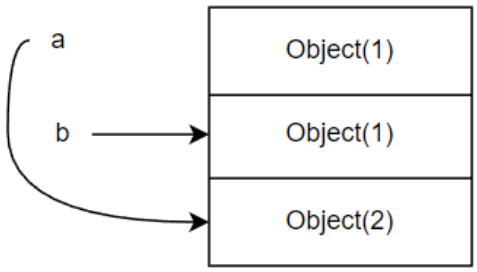


Figure 6: Line 3 of the example code using CVM1. Based on figures from Rørnes et al.

CVM2

The second subcategory of CVM, the value of the variable a is changed at the memory location that a refers to after line 3 in the example code has executed as shown in Figure 7. This mental model is the least complete, or the least correct mental mode. When using CVM2, the answer will be indistinguishable from CVM1 based on the end value. [22]

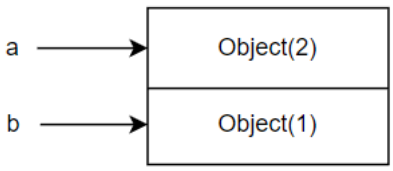


Figure 7: Line 3 of the example code using CVM2. Based on figures from Rørnes et al.

2.2.3. Copy Reference Model (CRM)

The copy reference model is the correct model to use in Python. If a programmer have the correct model, the answer should be correct every time. There are however two different sub categories of CRM[22]. As the different sub models of CRM give different results, it is possible to tailor the questions to identify the subcategory in the questionnaire. There is likely more than two sub categories, mapping new subcategories are not the priority of this thesis and as such only two will be considered. For both models, the code example from Figure 4 will act as shown in Figure 8.

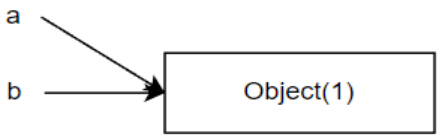


Figure 8: Line 2 of the example code using CRM. Based on figures from Rørnes.

CRM1

The valid copy reference model copies the reference to the object contained in the variable. [21] As such b points to the object in the variable a, not a itself. As such, changing the object in a does not carry over to b. At the end of the code in Figure 4, there are two different objects in a and b as shown in Figure 9. This is because the object contained in a in line 1 is after line 2 referred to by b, therefore the original object of a is stored in b.

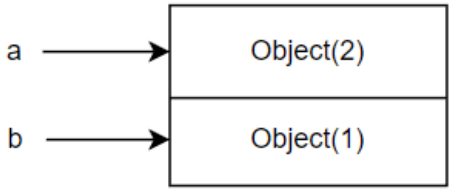


Figure 9:Line 3 of the example code using CRM1. Based on figures from Rørnes.

CRM2

The invalid reference model copies a reference to the memory location of the variable, not the object contained in it.[22] In line 2 in Figure 4 variable refers to the variable a and not the object in the variable a. By the end of line 3 both a and b both points to the same object. The reason is that the

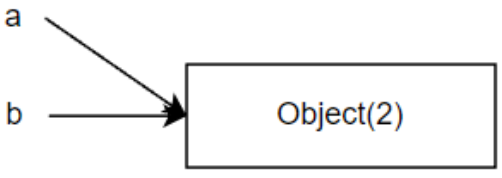


Figure 10: Line 3 of the example code using CRM2. Based on figures from Rørnes.

contents of a is changed at the memory location, which means that the object *b* refers to also has changed. This change is shown in Figure 10.

2.2.4. Models for functions

In Rørnes et al [22] there are mentioned three main categories of mental models for functions and in the full thesis [21] there are mentioned five mental models for functions. The use of the mental models CVM and CRM does not guarantee that the CVM and CRM will be used for parameters and functions.

```
1: list1 = [“First value”, “Second value”]
2: list2 = [“Value1”, “Value 2”]
3: def change(arg1, arg2):
4:     arg1 = arg2
5: change(list1,list2)
```

Figure 11: Example code based on examples from Rørnes et al and Rørnes master thesis.

The mental models FCOP and FOBJ are introduced in Rørnes et al [22]. Figures explaining the mental models will be using the code from Figure 11, that is based on the code examples in Rørnes et al[22] and the original example in the thesis [21].

2.2.4.1. Copy Value Model

The FCOP mental model is based on the copy value model where a copy of the references are transferred to the local scope as seen in Figure 12. The changes done to these local values will only affect the local variables as seen in Figure 13. The FCOP model is for simplicity called CVM for functions in this thesis. The FCOP model behaves as a CVM would and copies the values, a characteristic of the CVM models.

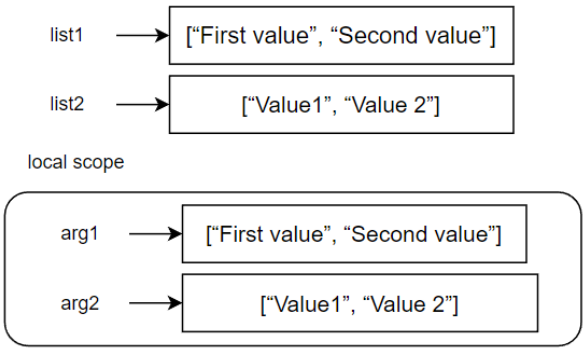


Figure 12: The parameter passing with CVM for functions.

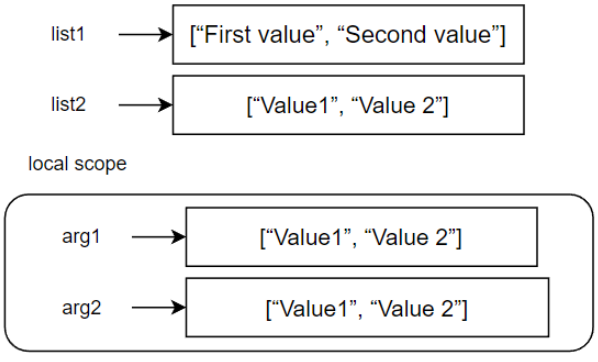


Figure 13: The changes to the local variables using CVM for functions.

2.2.4.2. Copy Reference Model

CRM1

The FREF model[22] is based on the correct mental model CRM1 for functions. Using this mental model, the reference is passed with the arguments as shown in Figure 14. The changes done to the variable act just as they would for any reference as shown in Figure 15. Unlike the CVM model where the change only affected the local lists arg1 and arg2, the append affects the object that arg1, arg2 and list2 refers to.

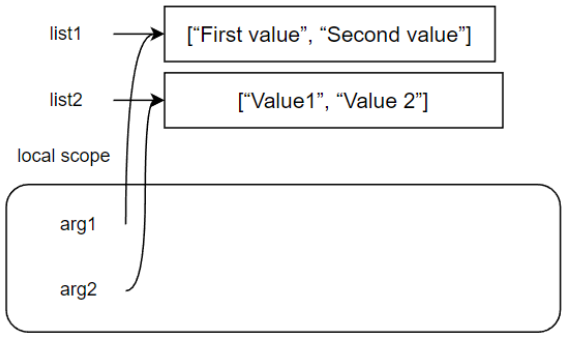


Figure 14: Parameter passing using CRM1.

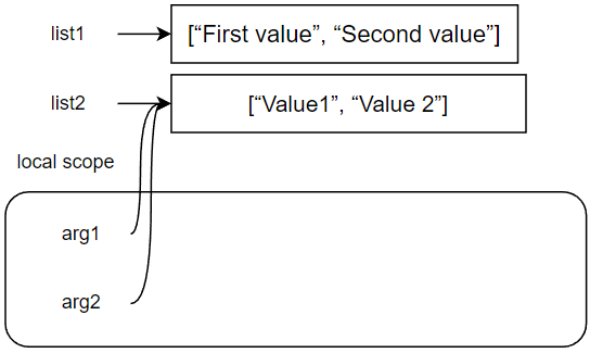


Figure 15:The changes to local and global variables using CRM1.

CRM2

The use of FOBJ[22] correlates closer to the use of a CRM2 model for functions. In Rørnes[21] thesis, the explanation is that the objects are passed as the arguments. The use of FOBJ will pass the arguments like shown in Figure 16. The changes from the function change will then affect the global environment like shown in Figure 17 if the of the CRMs are used for the reference variables. If one of the CVMs are used, the use of FOBJ will act as seen in Figure 18.

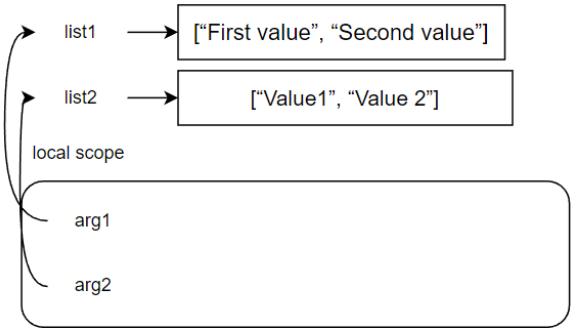


Figure 16: Parameter passing using FOBJ.

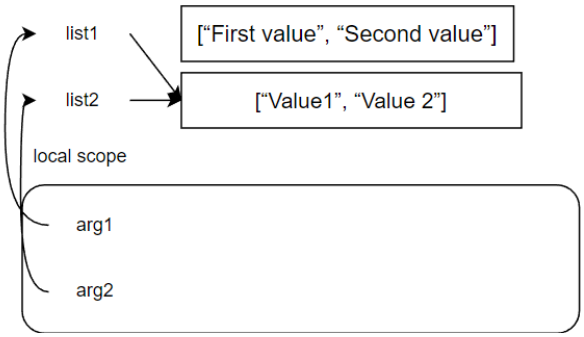


Figure 17: The changes to local and global variables using FOBJ with CVM.

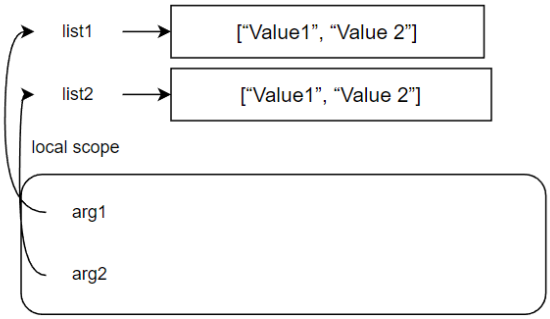


Figure 18: The changes to the local variables using FOBJ with CVM.

2.2.5. Patterns

In Rørnes thesis[21] the three mental models from the article is replaced by five mental models MS1-5. Rather than threating these models as separate models, it can be more beneficial to view these models as patterns. The correct model MS1 from Rørnes thesis can be seen as the application of CRM1 for both variables and functions.

The use of one model for both functions and variables are the use of a singular mental model. Using a singular mental model is the use of more than one mental model, the use of more than a one mental model is best described in patterns. An example of this approach is describing MS4 as the use of CRM2 for functions and CVM for variables. Making the description more applicable to a teaching situation.

2.2.6. Development of mental models

Henry and Dumas conducted a study focused on mental models with variables focusing on when references are learned by the students. The study was focused around one pre-test, and three test periodically from the students started learning programming. The last test was conducted after the participants had some practical experience with variables. Scoring one

point for the correct mental model and zero for the wrong mental model [5]. The study from Henry and Dumas use the mental models found by Dehnadi [2]. The model from this study relates less to the use of references and cover a more general model for programming. Since the models used in Rørnes [22] thesis is more focused to references and cover more specific cases, they will be used as a basis instead.

2.3. Notional machines for teaching references

The concept of notional machines (NM), although existing from 1970, it has only been popular since the start of the 2000's.[4] A NM is a pedagogic tool to assist with teaching complex aspects of programs or programming. Not all information is relevant in order to teach a given function or syntax in a programming language. Therefore, a NM focus at some information at the detriment of other information that can be taught later. It is in effect a model in varying degrees of accuracy in order to be simple enough to learn, but comprehensive enough to be useful [4]. A good example is the external representations of tracing made by Pythontutor.com. They give a visual representation to how objects act in Python.

2.3.1. The notional machines used to teach variables and references in IN1000.

This subchapter lists the various notional machines found in the IN1000 lectures and is expected that the participants come across. All quotes are translated from Norwegian to English.

NM1: "A variable is a given name that represent a value" [9].

NM2: Instructional code examples from the lecture. All code examples used during the first lecture falls under NM2 [9].

NM3: "A named location in the computer's memory where a value can be stored." The NM is given the description that states it is the most accurate, but the least useful way to think about variables. [9]

NM4: "A named parking lot where a value can be parked." [9].

NM5: “A nametag put on a value.” It described as a useful metaphor that is popular for Python [9].

NM6: “A variable can be seen as a «box» containing a value – a whole number, True or False, or a float number.” The NM includes Figure 19 [8].

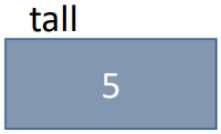


Figure 19: Figure from NM6.

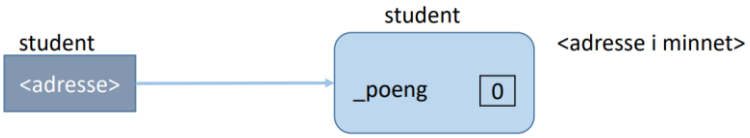


Figure 20: Figure from NM7 extending Figure 19

NM7: This NM builds directly on NM6. Adding that “Objects is not stored directly in variables – variables contain instead only the reference (address) to the object.” Adding Figure 20 to the figure in NM6. Explaining that references are often drawn as arrows between the variable to the object [8].

2.4. SOLO taxonomy

The SOLO taxonomy is a framework[28] to understand the response that a participant can give to a response. Generally used in education, SOLO focuses on how to interpret an answer from a student trough the explanation of a topic. SOLO is therefore better suited to think out loud questions, rather than written questions. [14, 23, 26]

The taxonomy defines five levels of understanding.

2.4.1. Prestructural response.

A prestructural response is the least advanced and sophisticated type of response. [14] A prestructural response lacs substantial knowledge about programming and how programs work and act. A prestructural response may vary greatly in terms of the code considered and the extent of the relation between the elements. The amount of code considered is called the width, while relations considered are called the depth in SOLO taxonomy. [14, 23]

2.4.2. Unistructural response.

A unistructural response utilize some but not all aspects of the code. A consequence of this partial understanding is what the Leeds working group called a “educated guess". Where the participant the partial understanding to arrive to the answer of the problem. As such a unistructural answer will likely contain a description of a portion of the code. [14, 23]

2.4.3. Multistrucutral response.

A multistrucutral response is where the participants understand the problem, while not having the awareness of the relationship of the different parts. As described by Lister et Al “the student fails to see the forest for the trees” [14]. The response is likely a line-by-line description of the of the code, summarizing each statement. Responses of this nature is likely not affected of the result, as such the answer may be wrong even when the logic behind it is false. [14, 23]

2.4.4. Relational response.

A rational response provides a summary of the actions and the purpose of the code. The novice integrates parts of the problem into a coherent structure and uses the structure to solve the task. The answers given by the response may, as the multistrucutral response, be correct or false. [14, 23]

2.4.5. Extended abstract response.

An extended abstract response is the highest SOLO level. The novice uses understanding to link the problem to a broader context. Commenting on that a given function will only work on a particular object type. [14, 23]

2.5. Tracing and its effects on students’ performance

Based on the BRACElet study[15, 16, 25] we can assume that the student’s ability to write and understand code correlates with the ability to trace code. This also correlate with the common assumption that a student who cannot read English cannot write English. Therefore, a student that have poor tracing skill, might consequently not understand references.

Tracing is an ability that is trained, and it is not always intuitive how to do it well. Different programmers can have vastly different styles and skill with tracing programs. The word skill is used as it is an activity that requires training to do well. A low-skilled tracer has no or low degree of ability to trace code, and a high skilled tracer has a high degree of proficiency and training to trace code reliably and consistently.

To use the SOLO taxonomy to describe the different degrees of tracers, a comparison between tracing and SOLO must be made, shown in Figure 21. The different degrees of skill, primarily a low skilled, medium skilled and high skilled tracer, is separated by knowledge and practice. It is therefore reasonable to assume that a person skilled in tracing one type of code is not necessarily skilled in every type of code. A degree of skill in tracing code cannot be a guarantee that the degree of skill is equal for a different piece of code or another coding language.

Tracing skill	SOLO level	Explanation
Highly skilled	Extended abstract	Can easily adapt to unknown code as the tracer has an understanding of how code works and can use this knowledge to tackle novel code. In other words, can predict how the code works, modify how the code works, and can understand and explain what the code does.
Medium skilled	Multi structural	Can apply existing knowledge to deduce what the function of the code is. The tracing is consistent and tackles similar pieces of code, similarly. Shows signs of one singular Mental Model for known issues.
Low skilled	Pre structural	Can write known code with assistance but cannot produce unique code. Needs to run all code and do not attempt to deduce the function of the code.

Figure 21: The overview of qualities of a high, medium, and low skilled tracer. Made from the results from the BRACElet study[15, 16, 25, 26] and the SOLO taxonomy.

2.5.1. The use of list to test tracing skill.

In [22] the use of lists has been used to test how well the participants understand references . Where most BRACElet studies use more complicated examples [15, 16, 25] to test the tracing ability of the students. The use of lists has more merit to its use, as the basic use of lists and variable reassignment is easy enough that a novice programmer will not have any misconception about the object. But it is complicated enough to identify the misconceptions about references and tracing ability of the programmer.

2.5.2. Tracing and mental models

The expectation is that a programmer that cannot reason for the choices made in the code, cannot trace well and therefore is reasonable assumption that they do not follow a consistent mental model. It is also reasonable to assume that improvements in their tracing skill, also will be better suited to converge on one singular mental model. The research done by Henry and Dumas [5] started a semester long study were more focus on the quantity, rather than why the participants changed their mental models. Their focus was to grasp when the shift from incorrect mental models to the use of a single correct mental model occurred. They found nine profiles that were based on students' understanding of references as a concept. They also include, after discussion with the teachers, that the student profiles would be more of a benefit if they focused on the students' weaknesses and not when they gain the correct mental model. The weakness in reference understanding can be understood as misconceptions and wrong mental models.

When do these wrong mental models occur and how do they form into one singular correct mental model, is yet unknown. In order to answer these questions, a semester long in-depth interview is beneficial to expose what the novice students think, and what influences their programming.

2.6. The Dunning-Kruger effect.

The Dunning-Kruger effect is often used when discussing a difficulty learning and is understood to mean that a person that lacks insight into a subject also lacks the ability realise the lack of insight. The effect is most prevalent on low performers where they overestimate their own abilities by thinking they perform as good as the average student [3]. An overestimation of their own ability is coupled with a lack of ability to reliably identify an individual with a high degree of knowledge in the subject. Lacking the ability to self-identify errors in their own approach can cause the student to overlook a better solution when presented with one. Thereby negating any intended learning.

The main issue is questions that the novices do not know enough to ask, called unknown unknowns. The unknown unknows are the opposite to the know unknowns where the novice knows enough to ask about the unknown [3]. Without knowledge of an unknown subject, a novice is limited to their ability to self-evaluate. As novices with a low competence in a subject is less identify the lack of competence, the novice is less likely to seek remediation

measures. The assertion that novices that lack competence in a subject can look up the answer can lead to the novice gaining low quality of understanding of a subject with high confidence in own abilities. The consequence is that new subject must be taught and not necessarily taken to be understood [17].

3. Research Methods

This chapter will present the research methods utilized during the study. The methods are based on the previous literature on mental models, precented in section 2. The focus was put on talk out loud exercises in interviews.

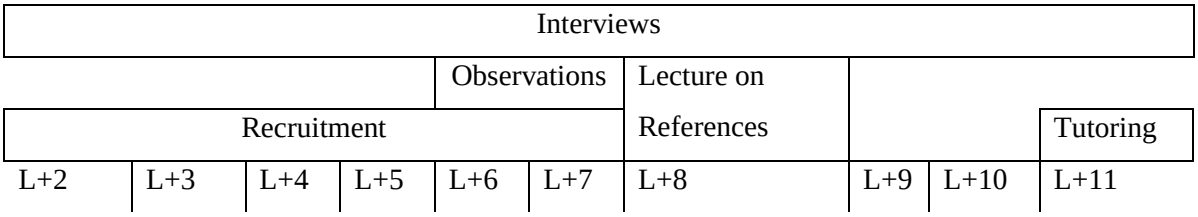


Figure 22: Timeline of interviews, observations and tutoring prior to the exams.

The participants were interviewed in relation to the course week, where week L+1 is the first week of the course. References are first introduced in week L+8. This is 7 weeks before the exam is held in week L+15 or also called E+0 the final exam is held. The interviews will be held prior to and after the lecture on references as shown in Figure 22.

3.1. Interviews

3.1.1. Planning the interviews

The interview form is based on the questions from Rørnes' thesis[21]. They were further expanded by applying the four different mental models found in the thesis to better facilitate the discovery of the different mental models and for each interview, a new set of questions would be created. The questions were palled to be continuously reviewed and improved upon.

The structure of the interview was to hand out a paper with five questions. Initially the participants were asked to explain what the code does in detail. After the participants had explained the code, the participants would be asked follow-up questions in order to better understand the reasoning, and any oddities that can occur during the interview.

A selection of five questions was chosen to not occupy too much of the student's time in the survey while still having enough questions to identify the different mental models. Making notes of the actions, questions, and explanations from the students. Reserving the follow-up questions until after the interview was completed, as to not impact the answers given during the initial part of the interview. The plan is then to analyse what mental models and what causes them, as to gain the ability to discuss these reasons and their development over time.

To reduce the learning effect, the answers to the questions will be reserved until the last interview. A flaw with this plan is the possibility for the participants to test the code themselves.

As the identity of the participants are known to the interviewer, the data that is collected is considered personal data. An application to NSD, the Norwegian Centre for Research Data, was sent in July and approved in early august with the project start 15. August 22 to the end of 23. The collection and storage of project data was deemed sufficient. All data was anonymized in order to be compliant with the NSD approval.

3.1.2. Recruitment

The recruitment of participants was done in IN1000 group lectures and problem-solving classes. A note was also posted on the Facebook group and Discord group for the institute with contact information. Participants were promised help in the preparation of the exam as gratitude for their assistance in the survey. This help was given in the form of tutoring on a one by one basis, see 3.3.

The initial goal was to recruit between 10-20 participants. A concern being that the participants would not be interested in such a long-term commitment, an emphasis was made to keep the interviews short. Initially two participants were recruited, they are given the unisex names of Sam and Kim for anonymity. After Kim stopped participation due to a lack of contact, Alex agreed to participate in week L+7 of the course and participated from interview 3 and out the duration of the course. Week L+8 the recruitment was halted as the subject of references would be taught in class.

3.1.3. Conducting the interview

The interviews were conducted individually and at a time that the students themselves chose. The participants were informed what was expected of them. Mainly that the answers are not the priority, and to explain what the thought process is in order to arrive at the answer. On the first interview, the participants was asked verbally if they are comfortable with an audio recording of the interview.

The interview itself is divided into two parts. Part one is the main part, they are given a form that consists of five questions. The participant is asked to answer and explain their thought

process through the form. Any unexplained actions or questionable decisions will be written down and taken into part two. Part two consists of follow-up questions. The follow-up questions concentrate on their actions and what made them do those actions. These questions are asked later to not disrupt or change the answers during part one, preventing the participants from answering too differently due to second-guessing themselves. The students will also be asked how they find the pace of the course and if it is difficult or easy.

As the priority is the mental model and not the ability to compare the answers themselves. New questions for interviews should add code that can easily be discussed, as to get better understanding of the reasoning by the students. Being difficult enough to trigger discussion and thoughts, however not so difficult as to cause confusion or that a lack of knowledge on the code can be confused the understanding of references. The first interview round proved this as the use of append had a clear influence on the answers of the participants and caused confusion that may influence the results. As such an emphasis to explain methods and functions was put on subsequent follow-up questions.

It must be assumed that the follow-up questions and the fact that they are monitored during the interview may cause an effect on the response in the next interview. Therefore, the interview is scheduled two weeks apart, as to allow the participants to fall back into their regular habits before the next interview. The hope is that this approach may mitigate some of the learning effects that will likely happen during the interview.

3.2. Observations

During the semester, a series of observations were conducted during group sessions and Friday Python, a weekly session where group teachers were available to assist with coding issues. The observations were conducted in two ways, if allowed, notes were taken, however sometimes a more informal talk and assistance with the mandatory assignments were used.

3.3. Tutoring in references

After the last interview a tutoring session was offered. Both participants that participated in the last interview accepted this offer. The tutoring was planned based on the answers given during the interviews, while also giving the participants the ability to decide where the focus will be. During the tutoring, a focus was put on problem solving, with 4 copies of the tasks sheet from each interview. The first session with Alex was prepared based on the lectures and

the feedback. Applying the parking lot metaphor NM4 and later Python tutor, an attempt was used to explain the subject in a better way.

When working through the different tasks that had been resolved in previous interviews, it became clear that NM4 was insufficient and therefore the use of only Python tutor was used with Sam. The use of Python tutor with explanation had a better effect. The second tutoring session with Sam started using Python tutor based on the previous tutoring session. Although Sam had only the method that worked for Alex. Alex had more issues that needed correcting, and the tutoring took up towards 4 hours including breaks, while the session with Sam took closer to one hour.

3.4. Group teaching

The results from the tutoring were intriguing prior to the follow-up interview after the Christmas holidays. To see if the results of the findings are applicable to a larger student group, the concept of a group teaching event was formulated. This group teaching will consist of a mentimeter survey, lecturing in what object and references are, tracing examples, and lastly a second mentimeter survey. The use of mentimeter was used for its ease of use, the fact that it is commonly used in other types of lectures, and it allows for readily aggregated data. This tool allowed for anonymous collection of the data and was known and easy for the participants to use. Gaining anonymity at the cost of the ability to see an individual student's progress.

In the initial mentimeter survey the questions are designed to give only the correct answer when the right mental model is applied. The tasks used will be simple enough to not cause any confusion, while difficult enough to only facilitate one mental model.

During the tutoring, the task that will be shown must easily show what the references does, while being difficult enough to be useful in a realistic setting. The goal of the questions is to cause questions and debate. As the questions are traced during the lecture, an attempt is made before each slide to get guesses from

```
1: word1 = "Hei Verden"
2: word2 = "Alfabetet"
3: norsk = [word1,word2]
4: word1 = "Hei! Verden"
5: word2 = "abc...æøå"
6: print(norsk)
```

Figure 23: Shows the first task in the pre-test.

```
1: superlotto = 5 400 000
2: def vinnere(bool, premie):
3:     if vinnere:
4:         return 0
5:     else:
6:         return premie * 2
7: lotto = [superlotto]
8: superlotto = vinnere(false,
superlotto)
9: print(lotto)
```

Figure 24: The second task in the post test. Testing the understanding of parameter passing while introducing a series of distractions to avoid recognizing the similarities with the first test in the pre-test.

the group to predict the code. Each guess is treated and asked why, followed by an explanation of why the conclusion is correct of false.

Tutoring a group cannot be tailored unless there is a screening process. As such, the tutoring needs to work for every variation of incomplete mental models, the focus will be if they have the complete mental model or an incomplete one. What mental model the participants have, can in some instances be estimated, however it is not a priority and is not measured in any way.

The following post-test was made to both be difficult in order to facilitate discussion and secondly to compare data with the pre-test. Since the participants had already seen a couple of code examples there was a need to have the questions sufficiently different so the participants would not just remember the answers. There was also a wish to have more ability to educate the participants of the test. The intended length was 2 hours, as this is an unreasonable time to expect students to participate in, as the survey and tutoring was voluntary. The 1-hour tutoring was compressed to 20 minutes, having the bare minimum. The post-test was also intended as a last tutoring opportunity to allow the participants to leave the survey with more knowledge than when they arrived.

As such the post-test was a harder than the pre-test and relied on any unknowns and questions to be asked prior to the answers being shown.

The test gave the participants a self-regulated amount of time to read and trace the code before being shown the possible answers.

An example on the difference between the easy pre-test to give the students the best possibility to solve the task by their own is task one of the pre-test (Figure 23) and the second task in the post test (Figure 24) that may at first look seem totally different, however is quite similar. The task in Figure 24 is a test if the value in the list is changed when the variable is changed. Being intended to be difficult, the students had good time to evaluate the task and ask questions before the possible answers were shown. Realizing that that the variable in line 8 does not affect the print statement in line 9 would make the task trivial. The same is true for the code in Figure 23 where the code in line 4 and 5 has no effect for the print statement in line 6. As such the two code examples both have the same reference statement. However, the code in Figure 24 contains a more complicated code structure, and therefore the similarities may not initially be apparent.

Upon reviewing the results, a large disparity was shown between the number of correct answers on task two on the post test shown in Figure 24 and task one in the post test. Task one, as shown in Figure 25, shows the ability to recognize what is changed. There is admittedly a large span between the difficulties in task one and two from the post test. This is both to have a comparable yet sufficiently different tasks, as well as to see if the amount of clutter in the task has an impact on the results.

```
1: int_list = [1,2]
2: bool_list = [True, False]
3: list = [bool_list, int_list]
4: int_list.append(3)
5: print(list)
```

Figure 25: The first task in the post-test. Testing the ability to append into an object that a list also points to. In limited degree testing some of the same aspects as task two in the pre-test.

4. Results & Analysis

In this chapter the notes and recordings from the interviews will be presented, as well as the notes from the observations and the results from the group teaching. In subsection 1 the interviews will be presented and analysed. In the analysis, only the mental models CRM and CVM are used, not FOBJ, this is discussed in subsection 5.2 of the discussion. Subsection 2 will contain the notes and analysis from the observations. Subsection 3 will present the individual tutoring while subsection 4 will present the group teaching. Each section will contain the present the results and then present the analysis. The reason they are precented together is to better show the progress made by the participants, as that is the main focus of the study.

4.1. Interviews

In this section each category of questions utilized in the interview will be presented and summarized with an initial analysis. As the questions for the follow-up interview are the same as interview 4 in week L+9, the follow-up interviews are included in this section. The section will end with a coherent analysis with the total development over time for the interviews, as well as a separate analysis for the follow-up interviews.

4.1.1. Variables

The use of non-mutable variables such as strings and integers were utilized to gain knowledge on how the participants, in week L+3, Sam and Kim, viewed the variables. The questions 1.1 (shown in Figure 26) and 1.2, focused on variables containing strings. These questions were only used in the first interview, and not any subsequent interviews. These questions was intended as a warmup and a simple comparison if the participants understood the role of a variable.

```
a = "Hello world"
b = "Hei verden"
a = b
b = "Hei vakre verden"
print(a)
```

Figure 26: Code from task 1.1 in appendix A.1.

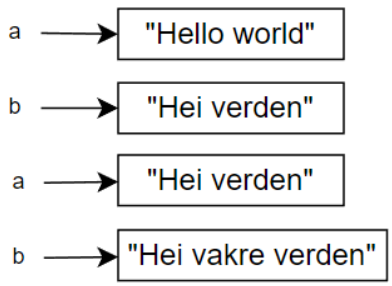


Figure 27: Tracing example based on Sam's explanation.

During the first task, both Sam and Kim explained the code as working from the top

down. Sam showed a generally good understanding of variables, what they are, and they do. When Sam was asked if a = b always equal b, Sam answered that “it stands still” as the value of a is only changed as a = is run. Therefore, a later change of b only affects b. Kim was unsure if “Hei verden” or an error will be printed in the terminal. When asked follow-up questions, Kim explained that reading the code led to some confusion and was now 100% sure that “Hei verden” is printed to the terminal.

During the second variable task, Sam trips up a bit on the math, getting confused in the calculations in the tasks. Sam ends up using the same reasoning as the previous task and arrives at the correct answer. While Sam deduces that the integer 5 is printed to the terminal, Kim believes that 2+3 is printed and not 5. Kim shows some insecurity and is unsure about the question and struggles a bit with the math.

The initial impressions noted after the first interview concludes that both participants are using the Copy Value Mental model. Another common trait is that both participants are very tuned into the code running from top to bottom. The belief that the line below the code cannot affect the code above was introduced during lectures and seems to be the guiding principle when handling variables.

4.1.2. Lists

Lists was tested throughout all the interviews, mostly utilizing a append function to change an object that also is represented inside another object. These tasks are categorized as List append tasks shown in 4.1.4. The second most used list task was the List flip tasks shown in 4.1.3. Two tasks that does not directly fit this pattern is the course task shown in Figure 28. The task style is used in interviews in week L+3, L+9, E+6 and E+10. These tasks were based directly on the tasks used in Rørnes study[21].

During the first interview in week L+3 the task 1.5 had two separate print statements, similar to 4.1 shown in Figure 28. In this task, Sam had the correct assessment with both print statements. The first print is correctly the same as the original list ["INF1000", "INF1010"]. On the second print Sam claims that oldCourses prints the original list and print(courses) prints [“IN1000”,”IN1010”]. Sam had noted that a change in the variables java and Python happened after the list was declared the change of the variables do not affect the list. However, they are changed after the list values are changed.

Kim deduces that the same task prints [INF1000, INF1010] on both print statements. When asked the follow-up question “what needed to be changed to get different prints?” Kim answers that a new list would need to be created. This is likely meaning that a new list oldCourses = [Python, java] would need to be created.

In interview L+9 Sam answers that the first print statement [“Hei verden”, “Alfabetet”] to the task shown in Figure 28. Sam struggled a lot and did not have a clear model. Therefore, the last two prints were that old_lang prints [“Hello world”, “The Alphabeth”] and lang prints [“Hei verden”, “Alfabetet”]. The correct printouts being first [“Hei verden”, “Alfabetet”] and then both print statements print [“Hello world”, “The Alphabeth”].

```
word1 = "Hei Verden"
word2 = "Alfabetet"
lang = [word1,word2]
word1 = "Hello World"
word2 = "The Alphabeth"
print(lang)
old_lang = lang
lang[0] = word1
lang[1] = word2
print(old_lang)
print(lang)
```

Figure 28: Task one from interview L+9,E+6 and E+10. Task 4.1 in the task set found in Appendix 4.

Alex answered that the first statement prints [“Hello world”, “The Alphabeth”]. And the two other printouts does the same. Alex clarifies that the list contains the variables themselves and not the value. Translated from Norwegian “The newest definition is the one that counts”.

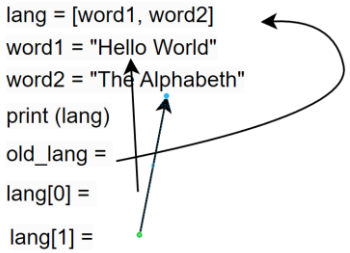


Figure 29: Participant drawing tracing task 4.1

During the follow-up interview in week E+6, Sam had a better understanding of the code. Showing more confidence in the task, as well as using terminology such as references when explaining the code. Sam also draws on paper the same way as during the tutoring (Section 4.3) shown in Figure 29. Sam also makes a point that old_lang and lang are the same object at the end of the code. Alex however, states that lang is equal to “the last”, and therefore lang will print [“Hello World”, “The Alphabeth”] and old_lang will print out [“Hei verden”, “Alfabetet”]. Alex pauses and corrects the answer to be the Norwegian print, [“Hei verden”, “Alfabetet”], for both. Since lang is redefined. During the follow-up questions, when asked why English then Norwegian was printed. Since we redefine what the variables contain, the last one matters. As such the last prints also are [“Hello world”, “The Alpabeth”].

Both Sam and Alex seems to understand how a list acts. Alex had some moment where the append function was difficult, and where the order was affected. However, both Sam and Alex had mostly a good enough understanding that how they view references were unaffected by their understanding of lists. The most difficult issue is realizing that list[index] = value sets the value to the corresponding index spot in the list.

4.1.3. List flip.

The use of list flip tasks were meant to separate CRM2 from CRM1 based on the answers. As such a variable redefined to be equal to another reference variable. The original reference variable is subsequently changed to another value as shown in Figure 30. The goal of the task is to discover the change, or flip, and point the change out in the result and discussion of the task.

```
a = [“values”]
b = a
a = [“values”]
```

Figure 30: Basis of list flip tasks.

During the third task in the first interview shown in Figure 32. Both Kim and Sam answer consistently with a CVM model. Both participants fail to recognize that a and c references the same object. Both Kim and Sam claim that a is not change, however Kim cannot reason for why. Append is a new concept for the participants, however it should be usable in the way it is used in the task. Both participants arrive to the conclusion that a is [1,3,4] and b is [1,2,4] when c is printed as [1,3,4,5].

```
a = [1,3,4]
b = a
c = a
c.append(5)
b = [1,2,4]
print(a)
```

Figure 32: The task 1.3 found in appendix A.1.

During the fourth interview in week L+9, Alex gave the correct answer, using CRM to arrive to the answer. Alex could not, however, reason for the answer or the way of thinking. Sam could reason a bit more about why the answer was given. Sam claims that after line three a, b and c will always be equal. Therefore, it will print 1000 on all three print statements. When asked why, Sam answers that since a = b = c then a = 1000 will change b and c also. Another answer is that if an initially is a String, the answer changes. If strings are used, then a, b and c will be different. Sam is highly confused about how objects work due to the clash with Sam’s pre-existing understanding of

```
a = [100,200,300]
b = a
c = b
b.append(400)
a = 1000
print(a)
print(b)
print(c)
```

Figure 31: Task five (task 4.5) from week L+9, E+6 and E+10. All tasks are available in Appendix A.4.

objects and variables and the new knowledge from the lecture on references. Saying that learning references earlier would be beneficial. Also including task like task 1,2 and 3 when students learn. Referencing the first second and third tasks from the task sheet shown in Appendix 4.

In the interview week L+11, both Sam and Alex went through the code line by line. In task one Sam answers that the printed list, is [“Minnie”, “Mikke”, “Langbein”] when b is printed to the terminal. Alex however state that the string “Andeby” is printed to the terminal instead. When asked why b prints “andeby” Alex says that it is equal to a and a is redefined to be “andeby”, so it throws away the list. The thought that the last definition is the one that counts was taught during the first weeks at the university.

During the follow-up interview, the same task sheet as in week L+9 was used. Sam traced the task rather quickly and correctly. Sam used the methods from the tutoring, similarly to the method shown in task one from the same week shown in Figure 29. Alex goes through the code line by line, the same way as in week L+9. Alex arrives to the answer that a is unchanged, while b and c are both [100,200,300,400]. During the follow-up questions, a mention was made by the interviewer on the rapid solution of the task, Alex then mentions that the variable a prints 1000 and not a list. When asked why b and c are equal when a change in the variable a does not affect the variable b and c. Alex explains that variables only work one way, when asked to explain why, Alex retracts the statement. The printout from c is then changed to [100,200,300].

In the tasks utilizing the flip questions, it was easy to differentiate the different types of CRM model used by Sam and Alex. One example of this were the task 4.5, shown in Figure 31, where the answer that a,b and c print the same correlates with CRM2, the answer that b and c print different results correlates to a CVM style model. The correct answer, a = 1000 and b = c, points to the correct mental model, the CRM1. We can therefore see that both Sam and Alex used predominantly CRM2 to solve the tasks before the tutoring. After the tutoring, Sam had retained CRM1 that was taught in the tutoring while Alex had not. We can also see that both participants are heavily affected by the NM of the code being “stupid” and working from the top down, not looking back based on the explanations.

4.1.4. List append.

The list append tasks mutate an object that has more than two variables that refer to it. This should result in both variables pointing to the mutated object. The difference between a append task and a list flip task is that the append task does not specifically differentiate between different CRMs by the answers directly. The tasks known as list append, were prevalent as they were easy to produce in a varied way. The tasks shown in Figure 34 and Figure 33 were also a basis for difficult and easy tasks by adjusting the complexity. This control afforded some predictability on how easy or difficult a task was experienced by Sam and Alex.

```
a = [1]
b = [1,a]
a.append()
print(b)
```

Figure 34: Basis of list append tasks, alternative 1.

```
a = [1,2,3]
b = a
b.append(3)
print(a)
```

Figure 33:Basis of list append tasks, alternative 2.

During the first interview, week L+3, both Sam and Kim showed some confusion regarding lists. Understandable as the concept is new, and they had not used it, only been taught in lectures. In the task 1.4 (appendix 4.1) both participants arrived at the same answer, where b and c were correct, however a was [[1],[2],[3]] instead of the correct answer [[1,4],[2],[3]]. Both participants used primarily the mentality that the code works from the top to the bottom. Kim was more unsure of the correctness of the answer than Sam, as with other answers.

The second interview had a majority of list append tasks. In all tasks Sam goes through the tasks line by line. In the first task, Sam originally answers that the answer will be “True”. The reasoning that the list is not declared a second time. During the follow-up questions Sam states that the values are copied. When asked to further explain why they are equal, the participant answers that the values in nyListe are [1,2,3,4] or [1,2,3,4,5] but is unsure what it is. Sam guesses it is [1,2,3,4], thereby changing the answer to “False” being printed. The way to get “True” printed would be to append the values to nyListe as well. When reading the second task, Sam originally conclude that a is not changed and a = [1,1,2,3,4], b = [0,1,2,3,4,[5]] and c = [5,6]. When asked why b is appended [5] and not [5,6], the reasoning was that when b is appended, c is still [5] and therefore only [5] is added to b. Equal to how a is not changed, Sam answers that b in the third task is not changed as well. When asked why a and b are not equal to each other, Sam answers that a and b are not equal to each other because the change happens later in the runtime.

During interview three in week L+7, Sam, and Alex both use the same general thought pattern. In the first task they both answer that a is unchanged and b is [10,20,30]. Sam answered in the follow-up questions that the only the values were carried over and stored in the variable. This is also the case for the second task, where again both Sam and Alex give the same answer to the task, using CVM on this task again. Alex shows less of an ability to reason for why, opting to answering, “it just is that way”.

In the fourth interview, in week L+9, both participants had been to the references lecture. Sam says that the end print has a and b equal to [0,2,3,4,5,6] and c = [5,6]. This was clarified to be meant as a and b = [0,2,3,4, [5,6]] during the follow-up questions. This answer points to the mental model CRM, giving a deviation from task 1. Alex misread the task, believing the variable c to contain a number, and was later clarified to be [5] not 5. Alex mixed a bit between CRM and CVM when explaining the task. In the end the participant followed a CVM model for the variable a and CRM model for b and c. Why was not clarified and Alex gave no reason, Alex knew however that the answer was correct. Upon clarification it was clarified that a = [1,2,3,4], b = [0,2,3,4,[5,6]] and c = [5,6] when the code has run. The final answer is consistent with the expected result using CVM.

During the fifth interview in week L+11, the last interview before starting to tutor the same week, both participants arrive to the answer that c = [[10,100],1000]. Sam notes that c is a list of lists and explains what is in it. Since it is a number, then everything changes, so in we add 100 and b is 1000 in the list in c. While Alex originally answers that [10,100] is printed. When asked to clarify what c is at the end of the program, Alex says that it is [[10],100] as it consists of the original a and b. When asked when it is printed the answer was after line 3, Alex answers that after line 5 it will be transformed to [[10,100],1000] and that is the list that will be printed.

In the follow-up interviews, week E+6 for Sam and E+10 for Alex, only Sam seemed to retain the lessons from the tutoring. Using the same task sheet as in week L+9, Sam traced the task quickly, arriving at the correct answer. When asked how to retain an unchanged copy of the list, Sam answers that you can copy the list to retain the values contained in it. Alex however misses that b.append(c) gives a list [1,2,3,4,[5]] and answers [1,2,3,4,5]. At the print statement, the list [1,2,3,4] is printed. In the follow-up questions the Alex says that the append only goes to the variable b and therefore a in unaffected.

Both participants has showed some growth in their understanding of references. After the tutoring Sam retained CRM1 while Alex reverted to the fractured understanding shown in weeks L+9 and L+11 and somewhat before that too. Alex struggled more with how lists work than what Sam did and might have had further problems based off that. While Sam moved towards a more relational explanation of what the code does, Alex did not. Sam had started moving towards a more relational explanation in the interviews prior to the tutoring and especially after. Alex did not have the same progression, showing a more unistructural approach to explaining the tasks during the end of the study. In some instances, Alex did revert back to a approach resembling a prestructural explanation.

4.1.5. Parameters

The parameter tasks originally were introduced to check if the participants has the same mental model for procedures and functions as they do for other code. The difference between a procedure and function is that a function has a return statement. Therefore, there are procedures that check for correct handling of parameter passing and the correct use of local and global variables. The functions do the same as the procedures, however they can also check reference variables on in a larger way as shown in Figure 35.

```
def legg_til_en(var):
    var.append(1)
    return var
x = [1,2,3]
y = legg_til_en(x)
print(x)
print(y)
```

Figure 35:Task 4 from the task sheet used in interview 4. The full task sheet is in Appendix A.4.

During the second interview in week L+5 Sam points out that there is no return value in the fourth task. As such no value is returned. To get a change of value a return statement must be given. If the nyListe was declared outside of the function and called on the function, the value will be stored in the procedure. For example, if a was called nyListe inside the procedure. Touching on the concept of references, but still clinging to the copy value model. For the fifth task the answer a=[1,2,3] was given initially. During the follow-up questions the participant clarifies that the correct result is a=[1,2,3,4] as liste in the parameter of the function in actuality is a. Showing a correct understanding of references. In follow-up question the participant identifies that nyListe could be an argument and the same would be true for task four.

```
def legg_til_en(var):
    var.append(1)
    return var
x = [1,2,3]
y = hent_tall(x)
print(x)
print(y)
```

Figure 36: Task 4 from the task sheet used in interview 3, week L+7. The full task sheet is in Appendix A.3

In the third interview at week three held in week L+7, Sam deviates from the answers given in the previous interview. The answer to task four (Figure 36) was that the value that was returned was 80. When asked follow-up questions the answer was changed to the list [1,80,3]. Given previous questions and the confusion that was shown when originally answering, some confusion was evident. Sam clarifies that y is equal to the function that returns the value of var that equals the value of x. Alex claims the function returns 80. Showing a lack of understanding on how lists work. That the x list is not changed points to a CVM model, consistent with task one and two. In the fifth task, Sam gives the correct answer and Alex does not. Sam shows understanding on how local and global variables act and understand that a reassignment of the local variable c does not give a change in the global variable c. Alex concludes that c = [1,2,3,4,5,6] and d is the same.

During the fourth interview held in week L+9, both Sam and Alex had been participating in the on references the previous week. For the third task, Sam answered consistently with CVM, while Alex answered consistently with a CRM. Sam noted that the procedure has no return value. When it prints a, it prints [1,2,3]. leggTil does something but does not store anything. Sam has therefore not developed the understanding of global and local variables since the previous interview. For the fourth task, Sam notes that x is unchanged, but y is the returned list [1,2,3,1]. Alex arrived at the conclusion that x = [1,2,3] and y = [1]. Alex was unsure about y, but sure of x. During the interview, this way of thinking was put under CVM as x is unchanged.

In the last interview before the tutoring, in week L+11, both Sam and Alex started to show signs of explaining what the code does and why, rather than reading line for line. For the first task Sam explained that the procedure adds a number to a list, but since it does not return anything, it will not affect the print statement. Adding that since it's not a return statement in the procedure, the new list is not stored anywhere. Alex had a different opinion. After explaining what the code does and then why it does it. Alex showed a correct understanding of how objects work. Explaining that the list was passed to the Procedure and the procedure added the value of the list. The print statement printed the changed list and therefore displayed the change. Alex did not show the same proficiency in the next task, as Alex was confused by the complexity of the task. After explaining what for i in len(list) does, Alex

deduced that the list a got transformed into [81,82,83] and this list was printed. Sam did the opposite, assessing correctly what the function does, however, fails to recognise that the function also changes b.

The follow-up interview was conducted in week E+6 for Sam and E+10 for Alex. Sam recognised and treated the local and global variables correctly. Sam had some light issues, but reasoned trough them and arrived at the correct answer. Using the logic that the leggTil(a,4) passes the object to the procedure, Sam reasons that the number 4 is added to the list that a point to. The issues with the task stem from the mental model of a procedure where everything in the procedure is deleted when it is done. Therefore, it feels wrong that something done inside a procedure can affect an object outside the procedure even when the object is passed as a parameter. When asked what will happen if the procedure contains list = instead of list.append, Sam knows it will not affect a, but cannot explain why.

Alex on the other hand goes through the code line by line, concluding that the print statement prints [1,2,3,4]. When asked why on the follow-up questions, Alex explains that a is passed as a parameter and therefore the append goes on the list a and is reflected in the print statement. For the last question utilizing functions, Sam quickly traces the tasks and reasons that x = [1,2,3,1] inside the function. Since there is a return statement in the function, the variable that contains the return value contains var. Since x was passed to the function and had been changed, and y = var since var is returned. Then x and y both points to the list [1,2,3,1]. On the follow-up questions the Sam say that a return statement makes the code easier.

Sam feels more confident when the function contains a return statement. When asked if y would be changed if there had been var += " " to a string instead, would still x be changed. The participant answers yes. What += does was explained to the participant, the participant then changes the answer a bit hesitantly.

Alex concludes after some thinking that x and y both print [1,2,3]. During the follow-up questions, Alex was asked what caused the issue, it became clear it was a misspelling in the task sheet, from a previous edition that was later corrected. When this error was explained, the answer was changed as x prints [1,2,3] and y prints [1,2,3,1]. When asked why x is not changed when a is in task 3, when confronted with this irregularity, Alex changes the answer that x and y both prints [1,2,3,1]. When asked why, Alex said that "I noticed the answer was wrong" [translated from Norwegian].

An interesting note when analysing the answers from the parameter tasks is the natural tendency toward CRM1. CRM1 allows for mutation with parameters, and it was something that Sam had learned during the course without explanation. Sam did however return to a CVM model after the lecture on local and global variables.

4.1.6. Loops

Loops proved to be one of the more difficult parts for the participants to understand, and therefore was not utilized as much as the other tasks. The reason was that the many moving parts made it difficult to predict. In order to test how well the participants understood references and not how well the participants understood loops, the loop questions were dropped from future interviews. The loop questions consisted mainly of one or two loops, where a value is appended for each iteration.

During the third interview, in week L+7, loops were introduced. When tracing a for loop, both participants struggle. Sam believes that the memory location, not the value is fetched in a for loop. This is in some way correct but fails to recognize that tall = 0 makes the variable equal to zero, not the value inside the object itself. Upon further questioning, this line of thinking was elaborated on, as Sam believes that tall is replaced with a[0] and a[1] and not equal to (tall = a [0]). Meaning that the conclusion is correct given the arguments but caused by a misunderstanding about references in for loops. On the second for loop, Sam uses a correct understanding of for loops. This results in b = [[1,0], [2,0]]. Given the previous misunderstanding, this was expected. Sam is therefore noted as using CRM when using loops. Alex uses CRM as well, changing the values in list a for the first loop. For the second loop Alex shows that the thought pattern is correct, the execution is not and concludes that a = [[0,1],[0,2]] and b is the same. Alex informs that the subject of nested lists and for loops is quite difficult. During the follow-up questions, Alex clarified that the append value should be behind the original value so b should be [[1,0],[2,0]] and reason a = [[0,1],[0,2]] is that tall = 0.

During the fifth interview, during week L+11 the second task containing loops was introduced. The task had an unfortunate error that was not discovered before the interview, where the loop appends to an integer and not a list. Since the error was introduced for one of the participants, the error was not corrected for subsequent interviews to be able to see the different responses from Sam and Alex. Sam discovered the error quickly before stopping to

make sure it is an error. Sam explains that you cannot add to a number in the list by saying "You cannot add something to that spot." [translated from Norwegian]. But had it said b instead it would work. You cannot append to a number. Alex spotted that there was an error in the for loop. Identifying that it was an issue, but not what it was. After some help the error was detected. This caused a lot of confusion for Alex. Alex had to some degree struggled with for loops previously. During the follow-up question Alex was asked why a[counter] + 3 change the value of a[counter]? Having claimed so previously. Alex answered that it was because it adds three into the first element. When asked if the same is the case for strings the participant stops to think and says maybe not. Alex fails to understand that the code produces an error without help, something that is understandable, but also trips on some more basic understanding of how variables work.

The for loops proved to be difficult when using the CVM and CRM2 models as the loop was difficult to understand. This might also be a consequence of the participants not having much experience with loops. In the case of Alex's answers, the answer [[0,1],[0,2]] proves that outside factors are a larger contributing factor than the understanding of references. And therefore, the task was dropped until the fifth interview in order to have comparability.

4.1.7. Summary

4.1.7.1. Interviews

When analysing the interviews, the notes in results are not raw data. The analysis is made on focusing on the primary factors in each interview and building on the notes and recoding of the participant. Focusing on the development and what triggered the development.

After the first interview, both participants are using the Copy Value Mental model, another common trait is that both are very tuned into the code running from top to bottom. The belief that the line below the code cannot affect the code above was introduced during lectures and seems to be the guiding principle when handling variables. It also seems that both participants also struggle with the append function. When using the form, a[1] = 1 it seemed like a Copy Reference Model was present, however a.append(1) was used the participants used a Copy Value model. It is therefore not clear if this is a singular mental model that allows for this behaviour, or the participants contain more than one mental model of references and variables depending on the situation. Both participants cannot reason about the code, read the code, and give mostly prestructural answers.

Kim is a bit more hesitant and unsure, while Sam is also unsure, but has more understanding of lists than Kim. Task 5 shows that Sam has an answer consistent with the CRM2 model. The follow-up questions did not manage to get any security of why the answer was given. The best estimation is that the participant has the notion that if a = b then a will always be the same as b.

In the second round, only Sam had the opportunity to participate. On most tasks the CVM model was used, but for parameters and functions a CRM model was used. Most answers were unistructural.

Sam could reason more on why the given answer was the correct one than during the previous interview. The current belief is that the participant has the CVM2 model. The interesting concept is that in Task 4 the participant floated the idea of that procedures can affect the value of a list passed as a parameter. However, later disregarded this idea. The participant returned to this notion on Task 5 and answered consistent with the use of a CRM model. When asked about the inconsistency in answering the questions, the participant said that Task 4 would store the answer if a variable with the name nyListe was declared earlier, based on the answer in Task 5 it is not unreasonable that this idea would transfer to the notion that nyListe could be a parameter as well. Essentially changing the mental model for procedures and functions to being a CRM model. Given the answers in earlier answers, a CRM2 model is more likely, however this cannot be determined based on the questions asked during the interview. If this is really a consequence of the model that was used and not an outlier, that implies that the concept of references are not that foreign even before the specific lecture is given. It remains to see if this is just an outlier or something representative.

For the third interview a third participant was recruited. The findings are supported by separate observation of student's groups working on the mandatory assignments. These students has the same difficulties and mental models found during the interviews in week L+7. Both participants had unistructural answers, Sam had however some multistrucutral answers.

What is interesting to see is that Sam has changed from a CRM to a CVM model for functions stating lecture in global and local variables. The answers are also coloured by a lack of understanding regarding for loops and other concepts. An interesting note is that Alex

showed a lack of tracing ability and also struggled more with the concepts than participant one.

Both participants struggle with for loops. Sam struggled with what exactly a loop does, leading to Sam believing that the first loop in task 3 made a = [0,0]. Alex struggle with understanding where in the list a append adds the value. Opting for a stack mentality where append adds a value at index 0. The underlying mental model used on loop 2 in task 3 is correct, however the wrong mental model for append causes the end result to be false.

When asked about functions Sam struggled with understanding parameters, now applying a CVM model as opposed to a CRM model as used in week L+4. The reason for this can be attributed to the notion of local and global variables. In task 5 the mental model gave the correct answer and was correct in order to test for global and local variables. The mental model was also closer to a CVM model on how the task was explained as well as task 4. Alex also had a CVM model in both tasks, the answer was however incorrect in both instances, showing a lack of understanding of global and local variables.

In the fourth interview in week L+9, the subject of references had been taught in lectures. Sam noted that it hard to unlearn what had been learned up until now. Sam also expressed a desire to have learned references from the start. Both participants were unsure about their abilities and felt less confident about their own understanding of Python after the lecture. Sam was more realistic in the self-assessment, while Alex felt more confident while being more often unsure and having a poorer understanding than Sam. Sam and Alex had more prestructural answers showing some regression, in large part both participants had some unistructural responses as well.

Sam struggled having one singular mental model, sticking more with a CVM model then a CRM. For functions, the misunderstanding that functions and parameters are totally insulated from the rest of the program and cannot make changes on the rest of the program unless a return statement is present is likely the cause of the CVM model for these instances.

Alex was a lot harder to understand why gave each answer or had a particular mental model. Following mostly the incorrect CRM model, sometimes following the correct one. Seemingly stuck to the NM that the code runs top down. The NM of code running top down in the way it works in the head of Alex prior to the lecture was an absolute rule, something that clashes

with object-oriented programming. Alex could in a lesser extent reason for why the stated outcome would be true, often stating that “it’s just that way”.

In the fifth interview week L+11, both participants still seem to have the same mental model as the previous interview. The participants are in large parts feeling that the topic is difficult and are unsure if the understating is correct or not. Alex is still largely confused by the code, but states when asked that the understanding of the topic is ok, but a bit more practice is needed.

Sam does a line-by-line explanation of the code for each task, mostly explaining what code is written, but not the larger function of the code. Still using the CVM model, most answers are consistent with the answers that were given in the previous interview. Lists and methods are trivial to understand, it is the understanding of references that causes the wrong answers in the tasks.

Alex had some of the same issues as Sam, going through the code line for line explaining each line, but not what the general structure does. A signal that both participants are more focused on each line, not the general function of the code. This causes problems as the need to check every line causes unnecessary confusion. This might also be a training issue, or a sign of lack of training. Alex also showed a lacking understanding of methods for lists and struggle with the mental model of the equal sign being the same in Python and maths. Confusing $a = b$ with $a == b$ and using the equal sign as an absolute meaning.

4.1.7.2. Follow-up interview

During the follow-up interviews, Sam was interviewed in early January, while Alex was interviewed in the middle of February. Attempts were made to hold the interviews in the middle of December, however this proved difficult to arrange. The date and time was decided together with each participant. Both participants got the option to hold a digital or physical interview, both choosing a physical interview.

Sam had retained most of the knowledge of the tutoring, the tutoring of the participants is shown in section 4.3. Using the tracing techniques shown during the tutoring. Each task was trivial and even if there was some second guessing, the answers were well founded. A difference in terminology and reasoning were also apparent, with more focus on the general function of the code. Using relational responses and the ability to follow the objects through

the code, instead of almost treating each line as an isolated piece of code as had been a habit before the tutoring, was the clearest indicator.

Alex struggled a bit more than Sam and had retained little if any of the tutoring received in early November. A clear indicator was that Alex, unlike Sam, used both CVM2 and CRM2 mental model. There was no clear understanding of what the code does, and the tracing was more ad hoc, and based more on each line separately than seeing the whole code. Alex also spotted a typo in the printed paper and did not ask for clarification, rather treated the function with a typo as a separate entity and concluded based on this.

After the interview, both participants were presented with the correct answers. Alex clarified that the motivation for studying had been low last semester and based on this had struggled with the course. The grade on the exam was also a failing grade. The issue for Alex seem to be more rooted in a lack of knowledge about programming.

4.2. Observations

In week L+6 and L+7, before the subject of references were taught in class, observations were carried out in exercise classes with students. During the observations a handful of students were studied, in total four groups totalling 10 students that were observed.

During the observation week L+6 two groups were observed. Both groups showed a lack of understanding regarding what the tasks entail. Both struggled with understanding the code and could not reliably read and understand the code that they were presented and wrote themselves.

Group 2 read the task more through than the first group, however, struggled to call functions in objects and did not read their own code. Group one did not read the task thoroughly enough and did not understand the code they wrote themselves.

Neither group showed any clear signs of CVM or CRM and could not deduce what model they used. Most of their problems stem from a lack of tracing and understanding the task, then coding concepts.

Second observation conducted week L+7 focused more on individuals, instead of a fly on the wall tactic a more direct approach was used and could identify if the individuals had a CVM

or CRM model. Out of three individuals one was CRM, and the rest was CVM. One of the participants, Alex, who presented with the CVM, was verified using an interview.

In the observations, most students that exhibited CVM rarely ran their code until they were finished with it and struggled with tracing their own code. The individual presenting the CRM model showed good code understanding and managed to read and predict the code before it ran.

There was also multiple students that said that the tasks given in the mandatory assignments were difficult to understand. None of the mandatory assignments so far have required a higher understanding of references.

4.2.1. Analysis of the observations

Out of these only one student showed a CRM model, while the 9 others displayed some sort of CVM when presented a question with two references. The feedback among the students that presented with CVM were equal to the feedback given during the interviews. Most students lacked the ability to critically analyse their own abilities. They therefore ran the code frequently to use the debug messages to pinpoint the bug without necessarily understanding why the bug got there in the first place.

The participants lacked the necessary ability to read their own code critically, so they used actively the group teachers and the debug tools in the terminals to find the bugs. Sometimes rewriting a piece of code that would be an easy fix. If the participants would have read the code correctly. Reference variables and arguments were often the cause of these relatively easy issues. The understanding of references were not the entire reason, issues such as a lacking understanding of what the task entailed also affected the lack of understanding. However, the inability to evaluate the function and behaviour of the reference variable caused the participants to not spot the issues themselves.

The observations proved useful to get an understanding of how the average student was positioned in relation to Sam and Alex. The observations were carried out in a more open setting and was primarily focused on the distinctions between CRM and CVM models, using around 20 minutes per student group. The behaviours and reasonings given by the observed participants of the group lessons and Friday Python match the behaviours and reasonings given by Sam and Alex.

4.3. Tutoring

4.3.1. Teaching Alex

With Alex, the first topic was variable declaration. What does a = b mean? Alex draws a mathematical comparison to variables being the same.

The fist notional machine used was the parking lot example from the lectures (NM4 from page 12). As this was presented as a good representation in the lectures, it was chosen as a less technical and more intuitive alternative. It was explained to Alex as:

If you a parking lot a and a parking lot b, and you drive a car into parking lot a called the integer 1. We cross park the car over b as well. We then change that the only car in parking lot a is the car called integer 3. What car is parked in b? The answer being the car integer 1.

The example falls apart when trying to explain lists. Using the example of a bus or a truck with cars in the back. When explaining modifying an object the example of changing colour or adding a spoiler to the car is used. The creation of a new object is explained as placing a new car.

After an explanation using the parking lot example, Alex tried anew with the task from the fifth interview. Alex, using this way of tracing and making sense of the tasks, showed a thought pattern reminiscent of CRM2. Alex struggled separating the variables from the objects. Walking through the code, an emphasis was made on separating variables and objects. Explaining as the object that the variable refers to, hence two variables can be referred to the same object. When appending to a list the object and not the variable is appended, meaning that a.append(argument) does not append to the variable a, but the variable a is a placeholder for an object.

Alex struggled with understanding how use object in list when the object is not referred to by a variable. Using an example of a trailer and the example of a friend group. The friend group consisted of: If var = object() means you are best friends, the fact that you redefine var as something else, only means that they are no longer best friends, the object can still be a part of a friend group like a list. The object, or friend, is still the same.

After running an example code on a laptop to verify the tracing examples, Alex recommend using Python tutor to show the code. The tool Python tutor had been mentioned previously leading up to the fifth interview, as well as during lectures.

Using Python tutor, and the task 2 from interview 5 the relationships in the list referred to by c becomes clearer. The concept that redefining what object b points to will not affect the list referred to by c.

When asked if a and b is the same after line 3 in task 2 from interview 4, Alex answered no. When asked to clarify Alex said that a and b were different because we made a change to the variable b. Alex showed a CVM thought pattern, saying the result at list a is [1,2,3,4] at the print statement of the task. After some help, Alex says that the list b is [0,2,3,4,[5]] at the end. Still clinging to the CVM thought model. Alex was allowed to draw own conclusion, with questions on what and why the code acts. As well as being informed that the result Alex arrives to is false. Alex is given the opportunity to discover the solution, and not given the solution when an answer is wrong. After several tries, Alex is asked what c contains. Alex then answers that c contains a pointer, a synonym for reference.

Alex manages to better explain what the code does using Python tutor to trace task 5 from interview 5. Alex also shows an ability to predict the code. When shown how the Python tutor trace the code, and explained what the correct mental model includes, Alex has a visible epiphany on how to trace code, as well as how the code works. Some more work needed to be done to learn how to slow down and realize what the different functions do.

Alex says that a[counter] + 1 increases the value in the variable counter with one. When asked what counter is, Alex answer 0. When asked if you can change 0, Alex says yes. When informed that you cannot change an integer, but you can change the integer in the variable.

In task 4 from interview 5, the result will be [1,2,3,4]. Because tall points to every element in the list. The participant is given a second chance to explain, Alex answers then that tall is 4 and liste points to the list that a also points to. We then append tall to the list that both a and liste points to. Therefore, the number 4 will be added in a. So, a is [1,2,3] and after 4 is added it is [1,2,3,4].

Alex says that the parking lot example now begins to make more sense. When asked what would be easier to be told how it works (tracing references) and then try individually. The parking lot example did not make sense until the example was shown in Python tutor. Alex

says that it would be interesting to know a bit about the different mental models in the lectures. Alex also asked some questions about own progress throughout the semester. Alex was informed that although the mental models were wrong, all mandatory tasks are passed. All the tasks in the interviews are designed to catch the different mental models. In the beginning, there were a lot less explanation. Today the participant gradually showed a better ability to draw own conclusions and reason for why.

4.3.2. Teaching Sam

Teaching Sam, started with going through the answers for the task list 5. On task 1 the answer Sam gave the correct result, but the thinking was that a list of strings acts different than a list of integers. Sam had previously been testing code with strings. In task 2 the answer was wrong but would have been right if the same method had been applied in task 2 as in task 1.

When Sam was asked if the answer of task 1 would have been different in the fourth interview and the fifth, Sam answered no. However, the answer was different, and it may have been forgotten due to it was around a month between the fifth interview and the eleventh.

In task four, Sam was possibly a bit confused about local and global variables.

Since the most efficient way in the previous interview was the use of the Python tutor method of tracing. This method was therefore used in the second tutoring. When starting the tutoring session, Sam was asked if it was better to trace on paper or computer, Sam opted for paper.

The tutoring started with tracing trough the different tasks from interview 5. When tracing task 1, great emphasis was taken on that we never use append the variable, we append the object in a, the object a refers to.

Sam then asked when the concept that if a = b then a always equal b applies then? The way Sam had understood that a = b always will be the same is that the variables will always contain the same. So, if you change a or b by the equal sign, you change both.

In order to correct this misconception, the difference between changing an object and changing a variable was explained and drawn on paper. The difference being that changing a variable using the equal sign, changes the object the variable refers to. However, if the object is changed using a method like append using variable.append(content) the variable is in fact

the object and therefore what the machine writes is object.append(content), as such, all variables referring to the object can also access the new content.

Sam then say that a great emphasis was put on that strings acted differently, because the way the strings were stored are different than with lists. To help correct the misconception, it was explained to Sam that the difference between a list and a string or int is that the list has methods to change the content of the list. Strings and integers does not have such methods and therefore we must make a new object to get a new value.

The colour example, in Figure 37, was used to better explain the difference between redefining a variable and changing an object. If there is an object colour that has two methods, one called color.get() that returns the colour in the object, and another called change() that changes the colour in the object. If we have the example in Figure 37 what would happen on line 5? The participant says that since we do not change b, c and b is still the same. After drawing up a external representation, Sam asks if we set variable b = colour(“Blue”) will c still be the same as b? The answer to that is no, because we have redefined what b is. Sam states this is a more understandable then how it was explained in the lecture.

```
1: a = colour(“Yellow”)
2: b = colour(“Yellow”)
3: c = b
4:
5: b.change(“Blue”)
```

Figure 37: The colour example

When Sam is asked “what is the difference between a.change("Gold”) and a = colour(“Gold”)?” Sam answers that there is practically no difference. This is not true, the difference is that a.change("Gold”) causes a change to an existing object, while a = colour("Gold”) causes the software to create a totally new object.

Sam then asks if the variable b defined on line 2 in Figure 37 does not exist anymore after line 5. The answer is that it does, it has been changed, it has not been redefined or deleted. So, if a is used in a list, when a.change(“Gold”) is used, the object in the list will be changed. If the variable is redefined to colour(“Gold”), only the object the variable points to is changed.

Sam asks why was it so important to know that strings acted so differently? The explanation is that you cannot change a string, a method on a string will always return a new different string.

If we add two lists, on the form variable = list + list, the value of variable will be a new list consisting of the contents of both lists. Operations are completed, so if you have 1+1, the variable does not contain 1+1, it contains 2. As it is faster and more logical to have 2 instead of 1+1. The same is true for lists. If you have a = list_one + list_two, any changes later done to one of the lists are not reflected in the variable a. Sam asks if a = a.append(1) will contain a.append(1), the answer is that a will contain None. As a.append(1) does not have a return value, the method call is completed before moving to the next step. Therefore, calling a will not continually add 1 to the list.

Another topic of discussion is list[index]. The get function in Python, called on the form list[index], allows direct access to the variable that the value of the index is stored in. Allowing for the refining of the value or the return of the value. Meaning that both a[0] = 1 and print(a[0]) are both valid arguments in Python.

Sam struggled with task 4.4, saying it was difficult to draw a figure following what it does. Originally answering that x is unchanged, Sam was told that legg_til_en(x) is the same as legg_til_en(var = x). The relation was drawn up in a similar way as the external reoresentations from Python tutor. Showing that appending var is the same as appending on x. Therefore, x and y points to the same object.

Sam ask how the list y can be changed using legg_til_en(x) without changing x. You need to make a new object. Either by copying the list using a function that iterate trough the list or using the .copy() function that every list has.

Sam comments that drawing the external representations is different to just looking at Python tutor. And feels that it is a useful tool. “It is one thing to see how it is in Python tutor and another to do it.” [Sam, translated].

Tracing task 5.5, solved a couple of issues regarding objects. It is mainly the concept of global and local variable causes confusion. Sam started to noticeably think more about objects and not variables. For the function we_love_the_eighties(list) calling a = we_love_the_eighties(a) are the same as calling a = we_love_the_eighties(list = a). Sam originally believed that the list, not the object that a point to, is sent with. Sam also believed that if the function has a return statement, the content does not matter. Sam also believed that an object in the function cannot change an object outside of the function.

Sam was explained that objects outside of the function can be accessed by a function, and local variables can contain the same object as a global variable. The difference between task 5.5 and task 4.4 is that the function in 4.4 returns var. This made Sam believe that task 5.5 act different as task 5.5 do not return the variable, rather a fixed value.

The crux of the issue was helping Sam understand in a = we_love_the_eighties(a), list refers to the object that also a and b points to. So, by changing the contents of list, we change the values of the list object that the variable lists a and b points to. Therefore, the line print(b) will produce the list [80,80,80].

After going through and tracing all tasks from interview 4. Sam feels confident in own ability and also show competence in the tracing from the tasks. Sam was more aware of the object and followed the objects and not the variables. Sam asks some follow-up questions regarding how to use a method from an object in a class. These questions were prompted by a mandatory task called game of life from the course. The question was answered by using the colour class from Figure 37.

4.3.3. Analysis of the tutoring

During the tutoring with Alex, the usage of the parking lot example from the lecture proved to be largely ineffective. The use of Python tutor proved more effective and was subsequently used to teach Sam as well. The parking lot example seemed to be too difficult to follow for more complex tasks. The example did not help even when solving several tasks. Using Python tutor was scalable and understandable for both Alex and Sam to follow. Alex struggled with for loops as well and seemed to have a decent grasp on both references and for loops after the tutoring. Between week L+11 and E+10.

When teaching Sam the tutoring lasted close to one hour, a drastic reduction in required time in relation to Alex that took close to 4 hours. Sam also had a better starting point than Alex, having understood most programming concepts such as loops and list prior to the tutoring. The reason Sam was not taught using the parking lot example was a fear to worsen Sam's understanding. Alex was more confused after the parking lot example, and the risk was to not improve this before Sam's exam. Therefore, the use of Python tutor was applied to Sam as well.

With many things in programming, simply seeing the use of Python tutor was not enough for either participant therefore practice was required. The tracing was done by hand on printed paper. Drawing on the paper. Both participants wanted to see an example before trying themselves and required a couple of attempts to understand both lists, loops, and parameters properly. The participants both self-reported a feeling of usefulness of the exercise and felt more confident and secure in their ability. This feeling was matched with an increase in ability by both participants.

As interview tasks from previous interviews were used for the tutoring, there is a possibility that Sam and Alex learned the patterns and not understood the base principles. With both Sam and Alex, an emphasis was put on teaching why the answer was wrong. One of the methods was not to give the correct answer, only explain why the wrong answer was wrong. As an example, if Sam and Alex would have answered 1+1=3, they would have gotten the feedback that the + sign adds two numbers together. For example, 2+2 is 4, if you have 2 and get two more, you get 4. So, knowing that, what would you say 1+1 is?

The use of Python tutor seemed to better allow the participants to sort their thoughts and visualize new knowledge. When the participants learned why a previous misconception was wrong, a visual guide was useful to help the participant find the correct answer. In Sam's case, it was more useful than with Alex. Sam used the same style of external representations as Python tutor to draw by hand and better understand the new topics.

4.4. Group teaching

The group teaching was based on the experiences tutoring Sam and Alex and the positive response from the follow-up interview of Sam. The group teaching had a pre-test, a PowerPoint talking about objects and references with tracing example and ended with a post-test structured as a quiz with time limit. The PowerPoint can be found in Appendix C. The participants had control over the tutoring and could at any point ask questions or clarification, also during the pre- and post-test.

4.4.1. Pre-test

The result from the pre-test is shown in Table 1. The questions, shown in appendix B.1, were made to be understandable, and there were no questions to the tasks during the pre-test. The data was collected using Mentimeter.

Question	Correct answers	Wrong answers
1	Total: 13 Group 1: 9 Group 2: 4	Total: 13 Group 1: 4 Group 2: 9
2	Total: 10 Group 1: 6 Group 2: 4	Total: 16 Group 1: 7 Group 2: 9
3	Total: 22 Group 1: 12 Group 2: 10	Total: 6 Group 1: 3 Group 2: 3

Table 1: Results from the pre-test during the course.

4.4.2. Post-test

The result from the pre-test is shown in Table 2Table 1. The questions, shown in appendix B.2, were made to be discussed, as they involved some more aspects than the pre-test. This was deliberate to make it more difficult for the students to memorise the answers. The data was collected using Mentimeter.

Question	Correct answers	Wrong answers
1	Total: 18 Group 1: 10 Group 2: 8	Total: 2 Group 1: 1 Group 2: 1
2	Total: 8 Group 1: 3 Group 2: 5	Total: 15 Group 1: 9 Group 2: 6
3	Total: 21 Group 1: 11 Group 2: 10	Total: 2 Group 1: 1 Group 2: 1
4	Total: 11 Group 1: 1 Group 2: 10	Total: 13 Group 1: 12 Group 2: 1

Table 2: Results from the post-test during the course.

4.4.3. Analysis

As seen in Figure 38 and Figure 39, the second group preformed much better than the first group during the post test. There may be two reasons for this disparity. The first reason may be that the second group asked more questions both during the explanation of references and objects, and during the post-test. As the tasks in the post-test was more difficult and complex, the participants could ask about any unknowns. An example question would be what a.append(number), the answer would be given as "the number is added to the back of the list a". The questions asked in the post test was particularly retained towards what variable = list + list does in question 4 of the survey. It is not unreasonable that the extra difficulty of the post-test caused some of the disparity.

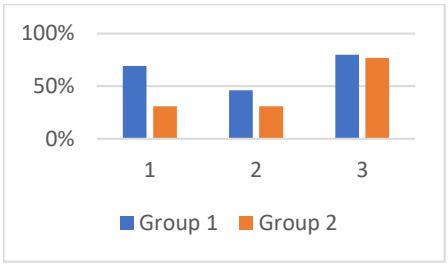


Figure 38: Percentage of correct answers to

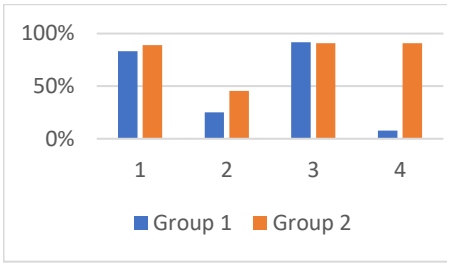


Figure 39: Percentage of correct answers to post-test.

The shortening of the group teaching caused the tutoring itself to be rushed, with fewer examples. And the last examples that the participants were supposed to solve before the post-test, were merged with the post test. The examples shown after the explanation was easy for the participants to follow, however the participants needed to be pushed to ask questions. The answers to the questions were often followed by further questions to explain the answer further. Showing that even when the participants had questions, they did not ask them.

The first group was less active than the second group during the examples. This is partly due to a extra focus on getting the participants to say what will happen at each line. The explanations from the participants were compared to correct mental models' explanation, these explanations were compared and pointed out why the explanation was correct or incorrect. The second group were pushed extra to give explanations, opting to not continue to the next slide before one was presented. This approach also had a domino effect causing more discussion as each wrong explanation was approach as a understandable way to think and followed with the aspect that was misunderstood and how that aspect actually works. For example, using CRM2 instead of CRM1, the concept of variables compared to objects would be explained. This was intended to explain the specific aspect of references that the participant had misunderstood.

An interesting note is that the first group scored better on average than the second group as shown in Figure 38. The second group went from average of 39% correct answers to 79% correct answers. While group one went from 66% correct answers to 51%. One of the reasons are likely that a lot of the participants got confused and rushed task 2 in the post test, as well as misunderstood how to add two lists in task 4 in the post-test. The second group asked what the result of adding two list were and the answer was defined as a new list, containing all the elements from both lists.

For a second group tutoring a series of changes would be made. Firstly, the post-test would need to be closer to the pre-test in difficulty. The tests should also be longer. As seen with both tests, there are some disparities in the percentage of CRM1 in both tests. By lengthening the tests to 5 questions or longer, closer to the quantity in the interviews, the end results should be less affected by difficulties belonging to one type of question. An example of this is task 3 in the pre-test, where a higher percentage gives the correct answer.

Another concern is the short timespan between the pre- and post-test. As seen with the interviews with Sam and Alex after tutoring. While Sam had retained the information after the tutoring, Alex had not. Even though Alex gave the correct reasoning and answers to the tasks that were shown at the end of the tutoring. The post-test is highly susceptible to remembering what the correct answers are, instead of arriving to them by their own.

5. Discussion

In this chapter the results and methods used will be discussed in relation to the previous research shown in chapter 2. Subchapter 5.1 compares the expected progression with what was observed during observations and interviews. Subchapter 5.2 discusses how to identify the misconceptions, and experiences from identifying mental models. Subchapter 5.3 discusses the first research question focused on how novices understand references in python. Subchapter 5.4 focuses on the second research question, how the understanding of references over time. Subchapter 5.5 focuses on the aspects that led to the participants to have a more complete mental model.

The actions from Sam and Alex have a lot in common, however they deviate in important ways. Sam's approach to references can more often correlate to a consistent model, while Alex applies an ad hoc approach more often than Sam does.

5.1. The expected progression

Prior to the interviews there was an expectation of how the participants would progress. Something that was in some ways met, and in some ways not. Proving the necessity for a standard way to recognize misconceptions of mental models. The methods was used to identify how novice programmers, primarily Sam and Alex, view references. What and how their understanding of references are shaped over time. Triggering the question, what causes the change? And how we can utilize this to help novices change towards a singular complete mental model.

Until week L+8, the expectation that primarily CVM will be present as the concept of references is not yet taught in lectures. This expectation was met with the first three interviews and the first two observations. All three participants showed a CVM model. Multiple students observed with CVM model and only one of the students observed showed a CRM model.

Why this student used CRM to solve the issue is unknown as the student did not want to participate in interviews. Observations were taken in group lessons focused on helping struggling students, so the presence of CRM might be higher for the course than is present for the group lessons. As the students are not explicitly taught references before week L+8 it would be beneficial to understand the thought pattern of a student that have arrived at the correct mental model. Had the student learned it previously or on his own?

The reason that only CVM was expected, was the notional machines used during the earliest lectures, and lack of instruction and need for references. Variables were introduced in the first week and was explained as NM1 “a defined name representing a value” and code examples in NM2. After which three different NMs was used to explain the concept of variables. In [9] the NMs NM3, NM4 and NM5 is shown with the explanation and their correctness. The emphasis was that variable is a container with a value, where the value in the variable can change over time. When reading and writing the code, a great emphasis is also put on the fact that the code is read from top to bottom. As the concept of references breaks in some degree with this notional machine, the use of references might not be directly intuitive for the participants.

“A named location in the computer’s memory where a value can be stored”	The most correct, but least useful example
“A named parking lot where a value can be parked”	A useful metaphor used in the curriculum
“A tag given to a value”	Useful metaphor popular in Python

Table 3: The NMs most predominantly used in the first lecture of variables, NM3, NM4 and NM5. With the usefulness as described in the lecture [9].

5.1.1. Wrong mental model or incomplete mental model?

Most students of the course have been able to complete the mandatory assignments, for those that never move past the CVM model, it is a testament that the “wrong mental model” is not hindrance, merely a restriction to making functional code. The correct mental model is the one that produce the right results all the time. Are all other models wrong? They still can produce functional code in the correct circumstances. A better term would be to say that the student has an incomplete mental model as the mental model is correct enough to produce the code at that stage. It is only when more challenging tasks is introduced that the model might produce the incorrect result or cause the student to make more complicated solutions to a simple problem. It might therefore to say the mental models are incomplete. As CVM works with objects if they only are contained in one variable at the time. CVM is therefore an incomplete model for mutable objects in Python such as lists. It is, however, functionally complete for immutable objects.

5.2. Identifying the misconceptions.

When interviewing the participants, a standardized approach to identifying mental models were used. A interview form was based on tasks from a previous study[21] and used as a foundations for talking out loud exercises. An attempt was made to use these tasks and the findings of the paper to create a set of tasks that produce the wrong result when applying the wrong mental model. The most effective of these were the list flip style tasks, as they allowed the answer to separate the CVM, CRM2 and CRM1. Something that the normal list append tasks did not do. The list append tasks were good tools to facilitate discussions, critical to talking out loud exercises. Therefore, the use of list flip style tasks were best utilized as a separation tool for easy separation between CVM, CRM1 and CRM2. These questions can prove useful for teachers of python to quickly gain an impression of how many in their class have understood references in python.

During the interviews, none of the students made external representations or drawings to explain except after when they were taught to do so. When explaining the code, the participants used a more step by step, explaining what code there was, not what the code did. As such the participants was more focused on the line-by-line execution of the code and not what the code does. This seemingly caused the participants to focus more on the individual lines of code and causing unnecessary confusion for themselves. It also made it more difficult to grasp the reasoning for the answers they cave. In other words, the reason for their mental model.

5.2.1. Mental model names, the previous research compared to observed behaviour.

Rørnes et al [22] use five different names for different mental models. For reference variables the concept of MVAL and MREF is introduced. In this thesis the names CVM and CRM are used for the same mental models. Furthermore, the listing of CRM1 is used as the correct mental model and CVM1 is the mental model closest to CRM1 in reasoning from the novice. This naming convention is similar in numbering, however different in the general name. The abbreviation CVM stands for copy value model, the use of MVAL was perceived as difficult to follow as it is not abbreviation, rather a shortened name. Therefore, as the abbreviations CVM and CRM were clearer and more intuitive, they were used instead.

Another three mental models were introduced for tackling functions. These three were compared to CVM and CRM, and no discernible difference was spotted based on the

explanation and illustrations. The use of FCOP is equal to how a individual with CVM would reason for the passing of arguments. The reason for this assumption is that the parameters in FCOP are copied over. A individual with the mental model CVM would likely assume that the value, hence copy value, is copied over to the local scope. Therefore the use of CVM can be seen as a continuation of the CVM for functions.

Similarly, the use of FOBJ can be seen as a continuation of CRM2, and is in this article seen as such. Both Sam and Alex struggled with separating the object from the variable when they seemed to be using CRM2. As the description of FOBJ from Rørnes et al is ".. FOBJ where the parameters are believed to be set to the actual objects passed to the function." [22], the confusion from the student could mean a more accurate mental model is that the value is changed at the memory location of the variable. This possibility is further strengthen by the findings of Vainio and Sajaniemi [24] where students are unable to use external representation. Often opting not to use them after being taught them, such as Alex in the follow-up interview. It is therefore most likely that the explanation that was given to Rørnes et al for the mental model shown in their article was caused by the participants confusing the variables with the objects. This causes the FOBJ mental model to be the same mental model as CRM2 for functions.

The use of CRM1 for functions, also called FREF from Rørnes[22], can be seen as a extension of the CRM1 for reference variables. As seen especially with Sam, the knowledge of Global and Local variables seem to be critical for fully understanding references as parameters. Based on the experiences of tutoring Sam and Alex, the difference between CRM1 and CRM2 for reference variables are the lacking separation of objects and variables. Equally the use of CRM2 for functions are caused by passing the variables and not references to the objects. For CRM1 the references are passed, this is also how the program handles passing arguments. It is therefore likely that the understanding of global and local variables causes can cause the use of CVM for variables instead of CRM.

The understanding of global and local variables can cause a user to use either CVM or CRM based on they believe that a local variable can affect the global environment. CVM does not allow for a function to mutate the global environment without having a return statement. If the function has a return statement, only the content of the return statement is returned and nothing else is changed. This is contrary to how CRM approach the subject of global and local variables. With the correct model CRM1 the arguments are references to the objects in

the global environment. Any changes to the objects will therefore be reflected in the global environment. This is contrary to CRM2 where the arguments are the variables themselves. Therefore, will any change to the variable affect the global environment. Using CRM2, redefining a variable will affect what the variable refers to in the global environment.

It is therefore possible to have two different mental models based on the understanding of global and local variables. The understanding of global and local variables affect what is passed as arguments. While the mental model for reference variables decide how the passed arguments are treated inside the function. It is therefore possible to have a CRM2 model for reference variables, while believing that a carbon copy is passed to the function. As seen with Sam in the later interviews.

5.2.2. Separating CVM1 and CVM2

Creating questions that gave a different result between CVM1 and CVM2 proved to unreasonably difficult as the models has no discernible difference exempt the mental understanding of where the value of the variable is stored. Because differentiating the different CVMs needed to be based upon discussions and follow-up questions, it was not prioritised as much as the difference between CVM and CRM. Separating the CRM1 and CRM2 was possible to differentiate as the use of them can give starkly different results.

Separating CVM1 and CVM2 can prove useful if the observation that Sam used CVM2 and then switched to CRM2 is prevalent in more students. Sam used originally CVM2 when asked follow-up questions. The belief was that the separation of CVMs were not important. It can be possible that the introduction of references caused a jump from CVM2 to CRM2. This is likely the case of the participants applying the new knowledge trough the existing mental model. When mental model was challenged with situations that could be proven false, Sam moved towards a more complete mental model.

5.3. How do novice programmers understand references and variables?

The most fundamental research questions in this thesis is the question of how novice programmers understand references and variables. This is further divided into the questions of how the novices understand how the code acts and how they perceive objects and references. The basis for answering these questions is primarily the interviews with Sam and

Alex. These interviews allowed for a deep dive into the mentality of the students that learn programming for the first time.

5.3.1. How do novices understand how the code acts?

One of the most useful insights of how novices and students think of code is the general phrase that “the code is stupid” and “it goes from the top down” as Sam and Alex explained during multiple interviews. Although not explicitly wrong, they indicate a more static understanding of how code works. And critically for this thesis, a lack of understanding of how references and objects work in the code.

Both Sam and Alex were highly focused on the fact that the code was “stupid” and started at the top and worked its way down. This combined with the concept of local and global code [6], locked the students to a copy value model and caused further confusion during the introduction of references. Students that were observed during the observation and during the recruiting process show a surprising ability to work around the issue and write functioning code.

5.3.2. How do novices perceive objects and references?

As both Sam and Alex used a variation of CRM2 before the final interview. This implies that both participants have understood the concept of references, however, does not know how to differentiate between the variable and the object. The findings in Rørnes et al[22] mention in FOBJ that the participants believe the parameters are the actual objects that are passed. While neither Sam nor Alex mentioned this model specifically, nor gave an equal explanation, they struggled with the distinction between object and variable. The model FOBJ is placed under CVM2 in the background because of this observation as well as the figures used in the article[22].

Even the participants in the group teaching had some variation of CRM2 or CVM, with only 56% having the correct mental model during the pre-test. These students had solved the mandatory tasks in the course and passed the exam. That begs the question, is it important to have the correct mental model if you can pass the course without it?

The ability to deconstruct the code and to find the reasons why the code fails, is an important skill among programmers. In the case of Alex, none of the lessons from the tutoring session

were retained in the follow-up interview. Alex and Sam had not worked on tracing tasks or made any extra efforts to maintain the understanding of references after the exam. Why then did Sam retain this knowledge, while Alex did not? One possible reason is that Alex struggled more than Sam with the general ability to understand code. Alex did, as many others that were observed, struggle with understanding the code, and subsequently debug the code. When asking if Alex struggle with understanding code, what does it mean to understand code?

5.3.3. What does it mean to understand code?

Understanding a field or a topic is not binary, there is some variation on how well the topic is understood. If understanding of code is defined as; the ability to predict the actions of the code, modify existing code, understand the actions of the code, and explain the code in an understandable way.

What does it mean to predict code? A programmer that can predict the code can read the code and predict the actions the machine will make, and the values that will be returned. As such they have an understanding of how the machine executes a line of code. References is a large majority of this understanding as the use of references is important to understanding how novel functions function.

When a code is modified, a programmer is required to know what the code does currently and is supposed to do in the future. Making code to tailor the existing code to the future purpose. As such the ability to predict code is important to writing the new parts of the code. If you have a black box such as in Figure 40, the ability to predict the code, is the ability to predict what comes out as output based on the input you give the box.

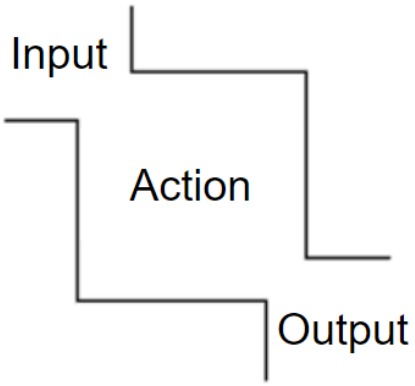


Figure 40: A example of a black box visualized in drawing. Input gets transformed by an action and returns an output.

Understanding the actions is understanding what happens in the black box, where only knowing the output and input is required to predict the code, understanding the actions of the code requires to know what happens inside the box. Breaking a big box into smaller bite size pieces. The smaller the pieces, the better the understanding. When the actions are understood, the actions

are no longer black boxes with an input and output. They consist of functions, variables, and purpose. The understanding of actions covers everything from how variables and references work, to appending to a list and more complex methods and functions.

The ability to explain code requires the ability to understand the purpose of the code and what the code does. Explaining code can be done by explaining what is on each line as the participants of the study did, or an explanation that sees what the code does. When the purpose of the code and why each line conduct their actions, a better understanding of the code can be achieved, and holes and bugs might become easier to spot.

With the definition of understanding code, the focus can then be put on why a novice would have issues understanding code. Some of the reason may be explained by the cognitive bias named the Dunning – Kruger effect. The Dunning – Kruger effect is a cognitive bias where people, in this case programmers, with limited knowledge or competence greatly over estimate their own knowledge or competence. Often used off hand to explain overestimation of own abilities. In the case of computer programming, it can be assumed to be where programmers fail to recognise that their understanding of the programming language or fail to predict the code.

A common trope in the programming community is the phrase “The code doesn’t work, why?” or “The code works, why?” as a comical way to explain a lack in ability to understand own code. The ability to read code, tracing, is a skill that requires practice. In the duration of the study, Sam, Alex, and the students observed during Friday Python show a lack of ability to predict their own code as well as understanding the actions of the code.

During the observations, a common reaction to the code not acting as expected is to find a work around, either by coding around the problem or brute forcing it. Brute forcing the problem would entail commenting out code, changing variables or rewriting pieces of code until the code acts as expected. The issue with this approach is that the students used it without understanding what caused the issue to begin with. Some of the students observed during Friday Python asked why the code did not work, not able to identify an error that had been pointed out in the terminal. It is therefore important to give the students and all programmers the ability to identify the limits of their own ability, and what remedies to seek.

5.4. How does the understanding of references change over time?

The second research questions is how the understanding of references changes over time. The question has three parts. Firstly, what causes the change? Secondly, are the changes triggered by lecturing and tutoring, or by the novice's own experiences? Thirdly, are they aware of the change?

5.4.1. What causes the change?

The first interview after the lecture on references in Week L+9 showed the effect of the lecture. Sam, having showed a greater ability to trace and read code, had a consistent mental model to this point. Sam had now many insecurities regarding the concept of references and felt that everything that had been taught previously needed to be unlearned. Alex had less tracing skill than Sam prior to the lecture, and now showed a more ad hoc approach to tracing had more models and could not account to one guiding principal.

Both Alex and Sam showed signs of transitioning between CVM and CRM in week L+9 after the lecture on references. As such, some confusion and disorientation on references is expected, especially having seen the confusion Sam had after the lectures of local and global variables. The clearest sign was task 4.5 in interview form 4 (appendix A.4) where both participants found the task itself easy. In this task Sam showed a clear preference towards CRM2 and Alex showed a clear preference towards CVM1. Alex felt confident that the concept was understood, however was inconsistent in the use of CVM1 and CRM2. While Sam felt that the lecture about references had uprooted any understanding of the concept.

After the fourth interview round week L+9, both participants showed a clear pattern of using a CVM for tackling parameters. For Sam this can be directly correlated with the introduction of global and local variables. Sam had prior to the lecture in global and local variables, used CRM1 to solve procedures and functions. Alex showed some of the same tendencies in week L+7 and L+9 as Sam. It is possible, but probable that the lecture in global and local variables caused both Sam and Alex to use primarily CVM for parameters.

In the tutoring after the interview week L+11, Alex was tutored first as the interview was the first one that week. Alex was furtherer confused when explained the parking lot example from the lectures. The parking lot example was explained as the variables are parking lots and when two variables point to the same object, the same car, it is equal to parking across two

lots. This analogy proved to not be good for explaining instances when a reference contains another reference. After trying several previous task sheets from previous interviews, Alex had deteriorated, and the use of the parking lot example was abandoned. Python tutor was used to explain references better and help Alex with forming CRM1.

The use of Python tutor to explain how references work and the difference between reassigning a variable and committing changes to an object worked incredibly well with Alex. It also helped with how immutable and mutable variables act as well as other aspects of understanding code that Alex struggled with. When Python tutor had been used, Alex started to use CRM1 and managed to explain the code. This allowed Alex to realise when errors were made and to correct them before testing the code in Python tutor. Alex also managed to solve problems with loops after this explanation, something that had been an issue for a long time.

Alex also had struggled with tracing for a long time. Not being able to explain correctly and consistently what the code did, nor why it did so. Most answers were a result of trial and error. In interview 4 and 5 (weeks L+9 and L+11) Alex did not have a consistent mental model and could not explain why each choice was made. The answers were based on previous experiences and novel problems caused great confusion and could sometimes not solve it without getting help. Alex had passed all mandatory assignments and felt that the current knowledge about the subject was complete.

Sam on the other hand did not struggle as much during the tutoring. This may have been a consequence of skipping the parking lot example for the Python tutor. It may also have been a result of Sam's higher understanding of how code works, not needing to be explained basic concepts and asking pointed questions.

The ability and utility of asking questions may also explain the difference between group one and two in the group teachings. Group two had a larger percentage of participants using CRM1 in the post-test then group one, going from 46% to 79% of participants using CRM1. The second group also was the group that asked more questions. The first group asked few questions and had a decrease in CRM1 percentage on the post-test relative to the pre-test, going from 65% to 54%. It is important to note that the post-test was harder to the pre-test. Group two had however a more consistent high percentage of CRM1 overall.

The change seems to be caused by new information or problems that do not correlate with the expected results. An example of this is the lecture in references. This lecture caused both and

Alex to revaluate their own mental model. The reason for this revaluation seems to be the introduction of new information that the current mental model do not allow for.

In the tutoring, the use of task that only give the correct answer with the correct model proved a useful tool for Sam and Alex to test their abilities and illustrate why the mental model was wrong.

5.4.2. Is the change triggered by lecturing and tutoring or is it derived from the novice's own experiences?

For both Sam and Alex, it seems that the change over time is directly affected by the lectures. The mandatory work and the other tasks only serve to refine and cement the mental model. No significant changes was found outside of the lectures.

This cannot be mistaken with a definitive answer what causes the change. One of the novices that were observed during the observations had the correct mental model prior to the lecture in references. What caused the change is difficult to know, it is however still possible that the novice learned the mental model either by own experiences or trough previous knowledge.

It is therefore difficult to say if the change to the correct mental model is triggered by lectures, tutoring or by the novice's own experiences. What can be said is that the lectures had a direct effect on the mental models used by the participants, as seen with Sam and Alex. And that a programming teacher teaching references cannot rely on the novice own intuition to learn references correctly.

5.4.3. Are they aware of the change?

It does not seem that the participants are totally aware about the different small changes. Such as Sam that gave answers following two different mental models in two different weeks for functions with return statements.

The larger changes that cause a larger, permanent change is noticed by the novices. Sam explained it as having to unlearn what they had previously learned.

5.5. What causes positive change toward a more complete mental model?

The third research question is what causes the positive change towards a more complete mental model? The question has two parts. Firstly, what actions can the students take? Lastly, what actions can the educators take?

5.5.1. What actions can students take to better their own understanding?

For the students for change the mental models they were dependent on the lectures. They did not question the why of programming sufficiently to encounter situations where the mental model was insufficient, or they did and worked around the issue. A key factor for a student is to trace the code when it does not work. Using python tutor or other tools to look at the code and why it does not work.

The key factor seems to be to challenge the mental models, and to investigate why. The investigation does not need to be independent; it can just as well be a question to the tutor. Questioning a tutor can be difficult however, as the novice does not understand what to ask. Unknown unknowns as Dunning calls them[3].

5.5.2. What actions can educators take to assist students with understanding references?

The use of visual aids such as parking lots and houses did not help Alex. Alex felt more confident about their ability to code, and the results from the tasks, without being more competent. Therefore, Alex had not acquired the tools to critically read and assess the code they were writing. These factors all lead to the belief that Alex saw the explanations and code examples using their incomplete mental models. This behaviour is similar to how the Dunning – Kruger effect makes it difficult for the low performers to identify a practitioner with more skill. This feedback chamber made the students overconfident in their own coding abilities as the examples seemed to verify their view, and the mandatory assignments did not challenge the mental models.

When shown drawings or other external representations of the code how the code acts. The novices show a rapid improvement in the understanding of references. Sam was more skilled in tracing than Alex, and the ability to change to a better mental model, and subsequently gain more knowledge in the same timeframe was clear. The participant was also able to

identify aspects of the newly introduced mental model, that caused confusion and misunderstandings.

Alex had a low tracing skill, especially compared to Sam, or low ability to explain the tracing. The general approach that Alex had, was representative of the majority of students participating in extra classes during the observations. Alex struggled with the parking lots example and ended up reverting to a CVM model. When the website Python tutor was used, Alex was able to gain a better understanding of references and of variables in Python. An interesting note is that Alex also saw a rapid improvement in tracing ability after tracing three tasks sets from previous interviews on paper with the assistance of Python tutor. At the end of the three task sheets, Alex was almost, if not, at the same skill level as Sam was after the tutoring.

Sam had tutoring later then Alex and saw rapid improvement after the use of Python tutor. Sam also asked more pointed questions, possibly aiding to the rapid improvement. For both participant the tracing of tasks had been highly efficient to give short term aid. And as the use of the tool worked well for both groups, it is likely a good tool to teach references in combination to actual tasks the students can use to predict the programs next step.

Two months later, both participants participated in a follow-up interview. Sam was interviewed first and showed that almost all the lessons from the tutoring had been retained. Sam also had committed to the use of tracing the tasks with external representations similar to the ones used in Python tutor. Most of the tasks were solved quickly and was nearly trivial to solve. Procedures, a subject that had been problematic previously, were the only subject that caused some sort of pause for Sam. Alex on the other hand had not committed in the same way to tracing tasks with external representations. Both tracing skill and general understanding of functions from append to how return statements had regressed to the same point as before tutoring. Alex also lacked a singular mental model, and solved most tasks ad hoc, with no clear plan. When asked why, Alex informed that the course had been difficult, while also lacking motivation. This had ended in that Alex had failed the final exam in the IN1000 course and would be taking the course a second time.

Why did Alex fail where Sam succeeded? A key factor is perhaps using external representation by tracing on paper to follow the code. Sam also had a better understanding of code in general than Alex. Alex had both issues with the code and the references. Therefore, Alex might have had issues other than references, these issues might have influenced the

understanding and caused the use of the ad hoc approach. As such the most important factors to teaching a novice can be summed as follows; First, the novice needs be aware of the existence of an issue. Second, the student needs to learn what causes the incorrect answer and what will cause the correct answer. The reasons that the complete mental model is the correct model to use. Thirdly the novice needs to train the use of the complete mental model in different types of code. The third stage can take many hours and requires that the novice learns tools to help with the process such as tracing.

5.6. Group teachings and lessons learned.

The group teaching was a success apart from two main things. Firstly, it is few results to calculate the significance of the results. Secondly, the last questions tried to fill too many roles. The post-test questions were meant to be challenging to get the students to ask about any unknowns they may encounter. The questions were therefore presented before the possible answers and were supposed to create discourse.

As the second test was based around both tutoring and testing, the extra difficulties required the participants to participate. As the second group had an 80% score on the post-test, most likely attributed to the increased participation and questions. Both groups would likely score more than 80% if the post-test had been tailored to just test in reference equally as the pre-test as it was originally intended.

Based on the experiences from the group teaching and from following Sam and Alex, some lessons can be taken and applied to future research. The first take away from the group teaching is the necessity for the participants in group teaching to participate as seen with group 2 and Sam. Although the results are not conclusive due to the issues with the post-test, the trends from group 2 and Sam show that student participation is an avenue worth exploring in future research. Further, the use of tracing can be a success factor for novices to better understand references. Sam had retained the understanding from the tutoring and used tracing actively during the follow-up interview. Alex did not retain the understanding of references from the tutoring and did not use tracing at all. As these are two novices and not a whole group, it is not possible to determine if the use of tracing is a tool novices should need to learn or if it is a results of Alex’s poorer understanding of coding in general. As Alex had issues using objects and some logic, and the issues were not solely centred around references.

Furthermore, the use of group teachings with pre- and post-test is a useful tool if the pre and post-test are more comparable in difficulty than was used in the group teachings conducted in this survey. The use of these group teachings early in the semester, prior to the lecture in references can be beneficial to testing out different ways for novices to learn and understand references.

5.7. Does tracing influence the use of a singular mental model?

Based on the findings from the Bracelet study, a low-skilled tracer will likely perform poorly in holding a consistent and good mental model. The finding that Alex, that struggled with tracing, also struggled with writing code, and needed to test every other line. This approach coincides with concept of Single Value Tracing[24]. It is clear that Alex struggle with more than just references, and also struggle with other aspects of understanding code. This observation that Alex struggle with coding and with tracing is consistent with the findings of the BRACElet study. Sam on the other hand was better at explaining the code, tracing verbally not tracing the code on paper. Sam had in other words the ability to reason what the code does and the fact that Sam did this while not presenting a singular mental model. This observation shows that the ability to trace code does not automatically give the participants the ability to understand the code. As such, the fact the participant can reason for the code, does not guarantee the correctness or completeness of the reasoning.

During the follow-up interviews, Sam used tracing on paper with external representations more actively during the follow-up interview. Since Alex did not draw on paper with external representations and also traced the tasks poorly, in relation to the performance during the tutoring. It is tempting to attribute Alex's lacking understanding of references to not tracing with external representations and the general low tracing skill. However, does the low tracing skill in tracing a cause the low understanding of code in general and particularly references? Or is the lack of tracing skill a symptom of Alex's lacking understanding of code, including references? The general impression of Alex during the follow-up interview was a general lack of understanding of what the code does, not references in particular. Understanding variables and subsequently references are one of the cornerstones of understanding what a program does.

The use of tracing techniques might highlight the actions of the code, it will not guarantee the understanding of the code. As seen with the students, a singular mental model is not

necessary to pass all mandatory tasks. The use of a singular mental model in regard to references is not a prerequisite to tracing. This does not mean that a singular correct mental model is unnecessary, as Sam traced well before learning the correct mental model, although the results were wrong. It does, however, remain possible that tracing using external representations such as Sam did during the follow-up interview, highlights the action sufficiently to realize deviations from the mental model and logical inconsistencies. There was no data from the interviews that could illuminate this question, and the tutoring were not set up in a way that can give any answers. Had the study been focused on teaching tracing and references to one group and have a control group that was not taught tracing and only references, the results would have displayed if references had an effect. This could be done similarly as the group teaching done in this study.

Both Sam and Alex had a singular mental model during one or more interviews, as such the use of tracing is not a prerequisite for having a singular mental model. It is, however, still possible that the use of tracing would lead the participants to better deduce and highlight their own shortcomings. Neither Sam and Alex showed any real progress based on their experiences outside of lectures and tutoring. Sam did some testing with strings prior to the interview in week L+11, however had not been able to realize why strings act different than lists. Sam had realized that strings act differently than lists, but not that integer, floats and boolean also act differently than lists.

If Sam had been able to trace the code, would Sam have been able to arrive to a CVM1 model for all objects? Or would Sam still arrive to the same conclusion that Strings act differently than other objects? As the survey was not set up to answer these questions, they are difficult to answer. A recommendation is to see if the teaching of tracing using external representation can affect if the novices learn references.

6. Conclusion

The goal of this thesis is to shed light on how novices understand references in python, how the understanding develops over time and what causes the novices to have a more complete mental model. The research questions to answer these questions as found in chapter 1.1 can be found in detail in chapter 5.3-5.5. The takeaways and findings of these discussions will be discussed in this conclusion.

This thesis have explored how Sam and Alex learned references and what caused Sam to learn to trace references. Using interviews, observations, tutoring and group teaching to find what caused Sam to use the correct mental model. As well as finding three key lessons to complete an incomplete mental model. The novice needs to both have the incomplete mental model challenged, as well as receive the necessary information to conduct necessary corrections.

During the interviews and observations, it became clear that most students struggled with understanding object and references, however, the same students struggled with finding errors in the code. The students followed in the survey, Sam, and Alex, was highly focused on the notional machine that the code was “stupid” and went line for line and cannot go back. This appaired to be the guiding principle for understanding code. When this understanding was challenged in week L+9, both students could not use the existing knowledge as a base for the new knowledge. Both Sam and Alex showed a lacking ability to critically evaluate their own abilities, causing a stagnation in their own ability.

There were two instances where the participants changed their mental models of references. First after the lecture on global and local variables, second after the lecture of references. As such the participants did not change the mental models based on their own experiences. During the interviews and tutoring it became clear that the reason they could not change the mental model themselves was that they did not realize the answer was wrong. They did not realize that the mental model was flawed and needed to change.

The three lessons from tutoring Sam and Alex were the three things that assisted Sam to get a singular and complete mental model. Firstly, the novice needs to know that something needs correcting. One way to let the novice discover that something needs correcting is to present a situation where the novice cannot arrive at the correct result with the current mental model. Secondly the novice needs to learn what causes the incorrect answer and what will cause the correct answer, the novice needs to be presented the reason. Thirdly the novice needs to train

the use of the complete mental model. The novice needs to learn the complete mental model. Therefore, to teach the complete mental model, the novice needs to discover the problem, be presented the solution, and learn the result.

This thesis is based upon the study following Sam and Alex. Tutoring Sam and Alex showed the effect of challenging the mental models with instances where only the correct mental model gives the correct answer, and the importance general knowledge of references to achieve a singular correct mental model. The group teaching with 28 students could not be used to find statistically significant proof as the sample size was too small.

Future studies are needed to gain statistically significant proof to the efficacy of this approach. These studies should focus on courses with tests like the group teaching, and further studies focused on before and after programming lectures about references. This will show if the three lessons from tutoring Sam and Alex are applicable on a larger scale.

7. References

1. Birtwistle, G.: SIMULA begin. Studentlitteratur, Lund (1979).
2. Dehnadi, S.: A Cognitive Study of Learning to Program in Introductory Programming Courses.
3. Dunning, D.: Chapter five - The Dunning–Kruger Effect: On Being Ignorant of One's Own Ignorance. In: Olson, J.M. and Zanna, M.P. (eds.) Advances in Experimental Social Psychology. pp. 247–296 Academic Press (2011). https://doi.org/10.1016/B978-0-12-385522-0.00005-6.
4. Fincher, S. et al.: Notional Machines in Computing Education: The Education of Attention. In: Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education. pp. 21–50 Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3437800.3439202.
5. Henry, J., Dumas, B.: Developing an Assessment to Profile Students based on their Understanding of the Variable Programming Concept. In: ACM Digital Library. pp. 33–39 Association for Computing Machinery, Trondheim, Norway (2020).
6. Horstmann, C., Necaise, R.: Python for Everyone. Wiley (2016).
7. Horstmann, C.S.: Big Java: late objects. John Wiley & Sons, Inc., Hoboken, New Jersey (2013).
8. Jensen, S.A.M.: Objektorientert programmering i Python. (2022).
9. Jensen, S.A.M., Sandve, G.K.F.: Variabler, https://www.uio.no/studier/emner/matnat/ifi/IN1000/modules/Week1/Main_modules/Variables/Variables.pdf, last accessed 2022/11/10.
10. Johnson-Laird, P.N.: Mental models and deduction. Trends Cogn. Sci. 5, 10, 434–442 (2001). https://doi.org/10.1016/S1364-6613(00)01751-4.
11. Kirkerud, B.: Object-oriented programming with SIMULA. Addison-Wesley, Wokingham, Engl (1989).

12. Krathwohl, D.R.: A Revision of Bloom's Taxonomy: An Overview. Theory Pract. 41, 4, 212–218 (2002).
13. Lehrmann-Madsen, O.: Object-oriented programming in the BETA programming language. ACM Press, Wokingham, Engl (1993).
14. Lister, R. et al.: Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. ACM SIGCSE Bull. 38, 3, 118–122 (2006). https://doi.org/10.1145/1140123.1140157.
15. Lister, R.: On the cognitive development of the novice programmer: and the development of a computing education researcher. In: Proceedings of the 9th Computer Science Education Research Conference. pp. 1–15 Association for Computing Machinery (2020). https://doi.org/10.1145/3442481.3442498.
16. Lopez, M. et al.: Relationships between reading, tracing and writing skills in introductory programming. In: Proceedings of the Fourth international Workshop on Computing Education Research. pp. 101–112 Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1404520.1404531.
17. Mahmood, K.: Do People Overestimate Their Information Literacy Skills? A Systematic Review of Empirical Evidence on the Dunning-Kruger Effect. Commun. Inf. Lit. 10, 2, 199–213 (2016). https://doi.org/10.15760/comminfolit.2016.10.2.24.
18. Masapanta-Carrión, S., Velázquez-Iturbide, J.Á.: A Systematic Review of the Use of Bloom's Taxonomy in Computer Science Education. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. pp. 441–446 Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3159450.3159491.
19. Maus, A. et al.: Rett på Java. Universitetsforlaget, Oslo (2011).
20. Murach, J., Urban, M.: Murach's Beginning Java with Eclipse,. Mike Murach & Associates (2015).
21. Rørnes, K.M.: Mental Models in Programming: Students' understanding of references in Python. (2019).
22. Rørnes, K.M. et al.: Students' mental models of references in Python. Nor. IKT-Konf. Forsk. Og Utdanning. (2019).
23. Simon: Assignment and sequence: why some students can't recognise a simple swap. In: Proceedings of the 11th Koli Calling International Conference on Computing Education Research. pp. 10–15 Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2094131.2094134.
24. Vainio, V., Sajaniemi, J.: Factors in novice programmers' poor tracing skills. ACM SIGCSE Bull. 39, 3, 236–240 (2007). https://doi.org/10.1145/1269900.1268853.
25. Venables, A. et al.: A closer look at tracing, explaining and code writing skills in the novice programmer. In: Proceedings of the fifth international workshop on Computing education research workshop. pp. 117–128 Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1584322.1584336.
26. Whalley, J.L. et al.: An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. (2006).
28. Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome), 1987, https://journals.sagepub.com/doi/abs/10.1177/089202068700100412, last accessed 2023/03/03.

Appendices

Appendix A

7.1. A.1 – Interview form Week L+3

Task 1.1	a = "Hello world" b = "Hei verden" a = b b = "Hei vakre verden" print(a)
Task 1.2	a = 2 b = 3 c = a + b a = 7 print(c)
Task 1.3	a = [1,3,4] b = a c = a c.append(5) b = [1,2,4] print(a)
Task 1.4	a = [[1],[2],[3]] b = a[1] b = [4] c = a[0] c.append(12) print(a)
Task 1.5	Python = " INF1000 " java = " INF1010 " courses = [Python, java] Python = " IN1000 " java = "IN1010" print(courses) oldCourses = courses courses[0] = Python courses[1] = java print(oldCourses)

7.2. A.2 – Interview form Week L+5

Task 2.1	hovedliste = [1,2,3,4]
	nyListe = hovedliste
	hovedliste.append(5)
	if hovedliste == nyListe:
	  print("True")
	else:
	  print("False")

Task 2.2	a = [1,1,2,3,4] b = a b[0] = 0 c = [5] b.append(c) c.append(6) print(a)
Task 2.3	a = [10,20,30,40] b = a a.append(50) print(a) print(b)
Task 2.4	a = [1,2,3] b = [4,5,6] def slaSammen(liste1, liste2): nyListe = liste1 + liste2 c = slaSammen(a,b) print(c)
Task 2.5	a = [1,2,3] def leggTil(liste, tall): liste.append(tall) leggTil(a,4) print(a)

7.3. A.3 – Interview form Week L+7

Task 3.1	word1 = "Hei Verden" word2 = "Alfabetet" lang = [ord1,ord2] word1 = "Hello World" word2 = "The Alphabeth" print(lang) old_lang = lang lang[0] = word1 lang[1] = word2 print(old_lang) print(lang)

Task 3.2	a = [1,2,3,4] b = a b[0] = 0 c = [5] b.append(c) c.append(6) print(a)
Task 3.3	a = [1,2,3] def leggTil(liste, tall): liste.append(tall) leggTil(a,4) print(a)

Task 3.4	def legg_til_en(var):
	  var.append(1)
	  return var
	x = [1,2,3]
	y = hent_tall(x)
	print(x)
	print(y)

Task 3.5	a = [100,200,300] b = a c = b b.append(400) a = 1000 print(a) print(b) print(c)

7.4. A.4 – Interview form Week L+9

Task 4.1	word1 = "Hei Verden" word2 = "Alfabetet" lang = [ord1,ord2] word1 = "Hello World" word2 = "The Alphabeth" print(lang) old_lang = lang lang[0] = word1 lang[1] = word2 print(old_lang) print(lang)
Task 4.2	a = [1,2,3,4] b = a b[0] = 0 c = [5] b.append(c) c.append(6) print(a)
Task 4.3	a = [1,2,3] def leggTil(liste, tall): liste.append(tall) leggTil(a,4) print(a)

Task 4.4	def legg_til_en(var):
	  var.append(1)
	  return var
	x = [1,2,3]
	y = legg_til_en(x)
	print(x)
	print(y)

Task 4.5	a = [100,200,300] b = a c = b b.append(400) a = 1000 print(a) print(b) print(c)

7.5. A.5 – Interview form week L+11

Task 5.1	a = ["Ole","Dole","Doffen"] b = a b[0] = "Minnie" b[1] = "Mikke" b[2] = "Langbein" a = "Andeby" print(b)

Task 5.2	a = [10] b = 100 c = [a,b] a.append(100) b = 1000 print(c)

Task 5.3	a = [1,2,3]
	b = a
	
	counter = 0
	for liste in b:
	  liste.append(a[counter]+3)
	  counter += 1
	
	print(a)

Task 5.4	a = [1,2,3] def leggTil(liste, tall): liste.append(tall) leggTil(a,4) print(a)

Task 5.5	a = [1,2,3]
	b = a
	
	def we_love_the_eighties(list):
	  for i in range(len(list)):
	  list[i] = 80
	  return 80
	
	a = we_love_the_eighties(a)
	print(b)

Appendix B

7.6. B.1 Pre-test for group teaching

```
word1 = "Hei Verden"
word2 = "Alfabetet"
norsk = [word1,word2]
word1 = "Hei! Verden"
word2 = "abc...æøå"
print(norsk)
```

```
a = [10]
b = 100
c = [a,b]
a.append(100)
b = 1000
print(c)
```

```
a = [1,2,3]
def leggTil(liste, tall):
    liste.append(tall)
leggTil(a,4)
print(a)
```

7.7. B.2 Examples from the PowerPoint used during the group teaching.

```
studenter = ["Ola", "Marie", "Kristoffer"]
program = ["Programmering 101", "Python"]
kurs = [studenter, program]
studenter.append("Lars")
program = ["Programmering 101"]
print(kurs)
```

```
a = [1,2]

def leggTil(liste, tall):
    liste.append(tall)

leggTil(a,3)
print (a)|
```

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum
a = [1,2,3]
print(listesum(a))
```

7.8. B.3 Post-test for group teaching

```
int_list = [1,2]
bool_list = [True, False]
list = [bool_list,int_list]
int_list.append(3)
print(list)
```

```
superlotto = 5 400 000
def vinnere(bool, premie):
    if vinnere:
        return 0
    else:
        return premie * 2
lotto = [superlotto]
superlotto = vinnere(false, superlotto)
print(lotto)
```

```
liste = [1,2]
def kryptograf(liste):
    crypt = [liste[1],liste[0]]
    return crypt
print(liste == kryptograf(liste))
```

```
familien_Eriksen = ["Lise", "Erik"]
familien_Henriksen = ["Emma", "Gustav"]
personer = familien_Eriksen + familien_Henriksen
personer.append("Jon")
familien_Eriksen.append("Krister")
print(personer)
```

Appendix C

Hva er et objekt?

- Objekter inneholder en eller flere verdier og kan utføre et sett med handlinger. Disse verdiene og handlingene er bestemt ut ifra klassen til objektet. En klasse er da oppskriften på et objekt.
- Et eksempel på objekter med mange verdier og handlinger er lister
- Et eksempel på objekter med få verdier og handlinger er tallverdier (int og float)
- De mest vanlige typene objekter vi har sett så langt er av klassen int, float, char, string og list. NB, Python behandler alt som objekter, i Java er det annerledes. Se forelesning om dette i IN1010.
- Verdiene i et objekt kan noen ganger endres med metoder. Metodene til objektet kan kalles på formen objekt.metode(argument). Et eksempel på dette er append i lister.

Hva er en variabel?

- En variabel har et navn og en verdi. I noen språk, slik som java, har den en bestemt type, dette betyr av den kan kun inneholde verdier av den typen.
- Verdien til en variabel i Python er en referanse til et objekt. Vi kan tenke oss at denne referansen viser til hvor i minnet objektet befinner seg.
- Vi kan endre hvilken referanse en variabel inneholder ved å sette variabel = ny referanse.
  - Når vi leser/tracer koden, så er det ofte nyttig å skrive ned hva koden faktisk gjør. I denne notasjonen er det nyttig å skrive variabel -> objekt eller variabel = -> objekt for variabel = objekt. Ettersom det er ikke et absolutt likhetstegn slik som ==, men en peker på et objekt. Vi skal se eksempler på det snart.

Hva er en variabel?

- En variabel har et navn og en verdi. I noen språk, slik som java, har den en bestemt type, dette betyr av den kan kun inneholde verdier av den typen.
- Verdien til en variabel i Python er en referanse til et objekt. Vi kan tenke oss at denne referansen viser til hvor i minnet objektet befinner seg.
- Vi kan endre hvilken referanse en variabel inneholder ved å sette variabel = ny referanse.
  - Når vi leser/tracer koden, så er det ofte nyttig å skrive ned hva koden faktisk gjør. I denne notasjonen er det nyttig å skrive variabel -> objekt eller variabel = -> objekt for variabel = objekt. Ettersom det er ikke et absolutt likhetstegn slik som ==, men en peker på et objekt. Vi skal se eksempler på det snart.

Spørsmål 1
Hva er forskjellen på kodeeksemplene?

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

```
a = [1,2]
a = [1,2,3]
```

Spørsmål 2

Hva er riktig resultat

a = 2

b = 3

c = a+b

a = 3

Er a == 5 eller a == 6?

Spørsmål 3

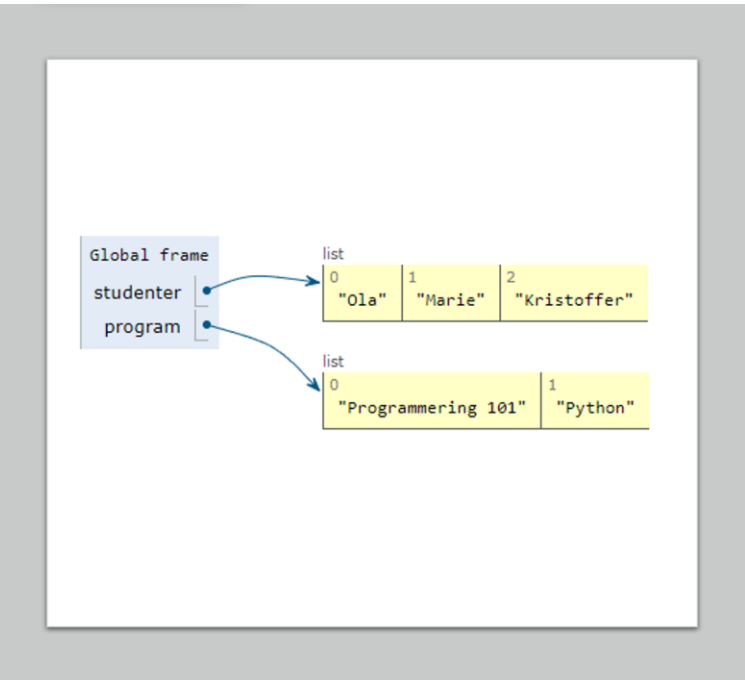
Hva er noen fordeler og ulemper referanser?

Kodeeksempler

- Vi kommer nå til å lese, eller trace, noen kodesnutter med lister, for løkker og funksjoner.

Lister

```
studenter = ["Ola", "Marie", "Kristoffer"]
program = ["Programmering 101", "Python"]
kurs = [studenter, program]
studenter.append("Lars")
program = ["Programmering 101"]
print(kurs)
```



Lister

studenter = ["Ola", "Marie", "Kristoffer"]

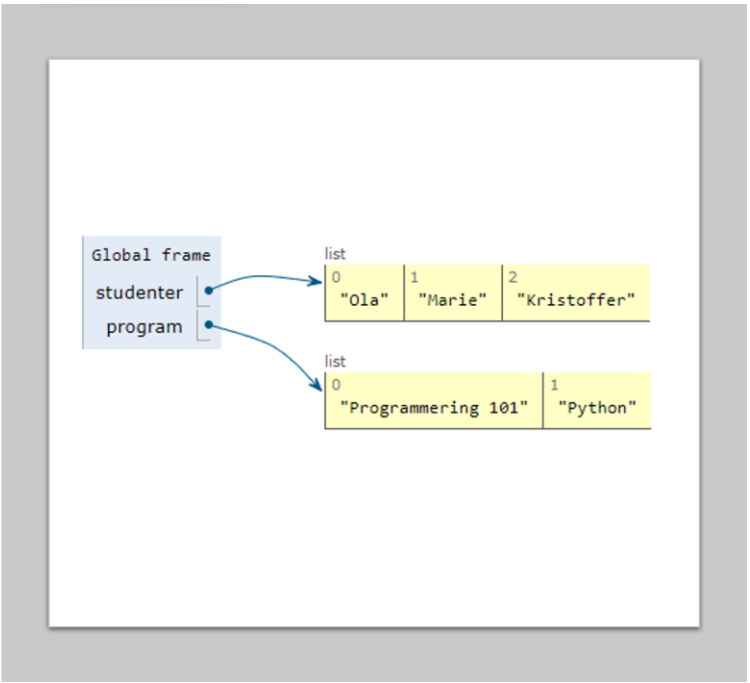
program = ["Programmering 101", "Python"]

kurs = [studenter, program]

studenter.append("Lars")

program = ["Programmering 101"]

print(kurs)



Lister

studenter = ["Ola", "Marie", "Kristoffer"]

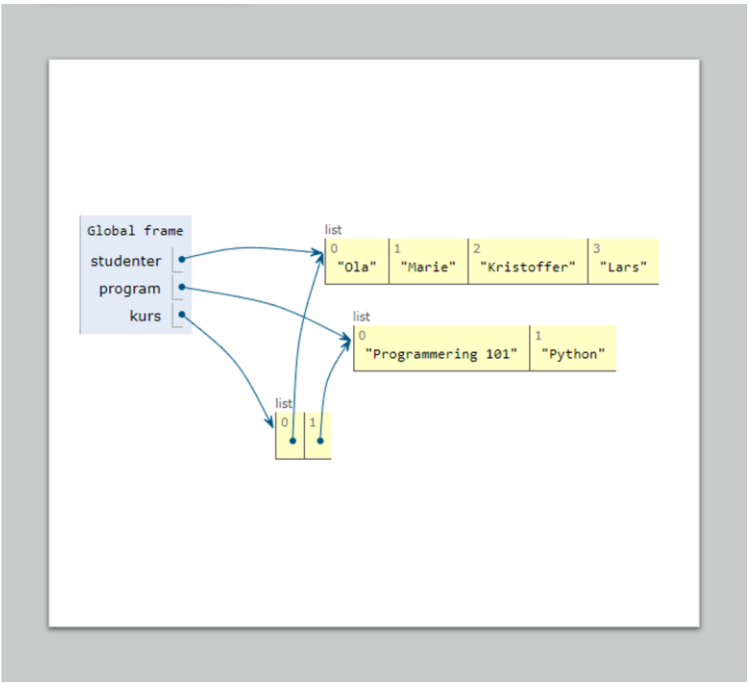
program = ["Programmering 101", "Python"]

kurs = [studenter, program]

studenter.append("Lars")

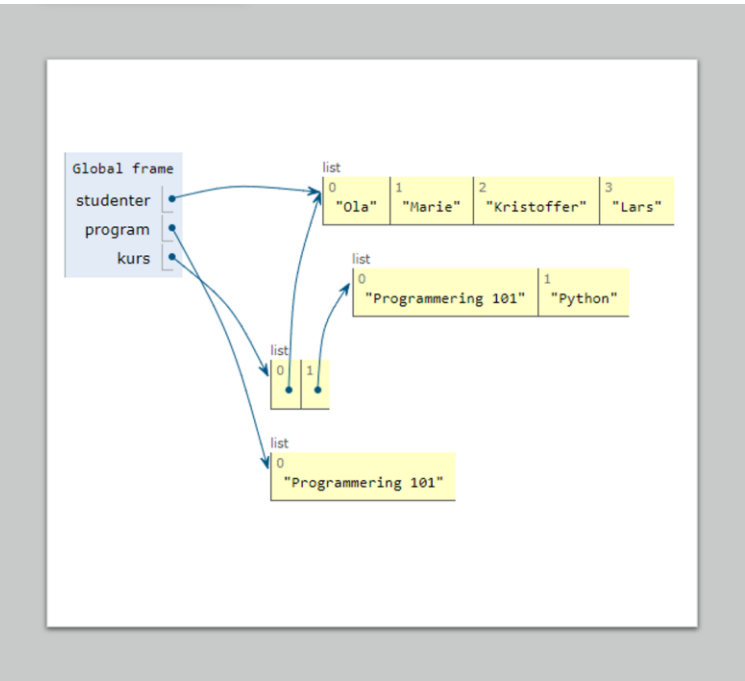
program = ["Programmering 101"]

print(kurs)



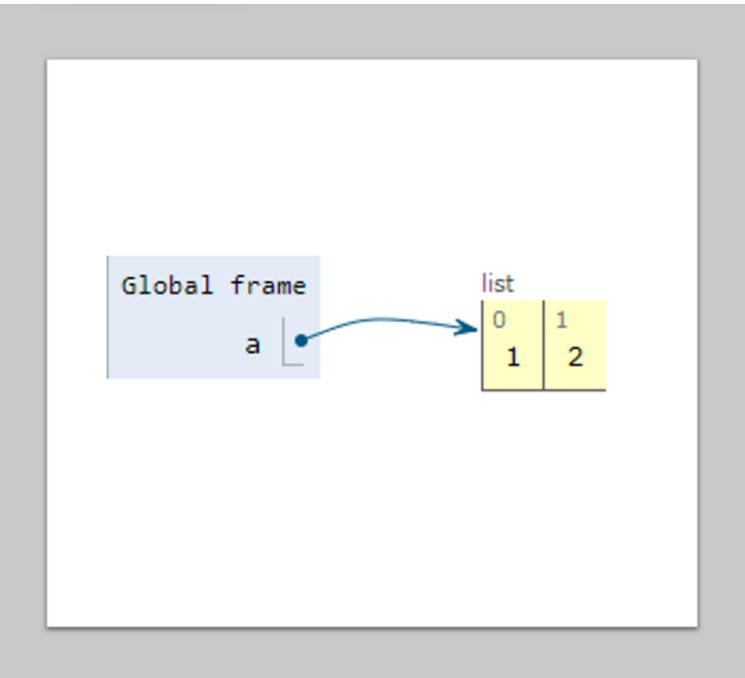
Lister

```
studenter = ["Ola", "Marie", "Kristoffer"]
program = ["Programmering 101", "Python"]
kurs = [studenter, program]
studenter.append("Lars")
program = ["Programmering 101"]
print(kurs)
```



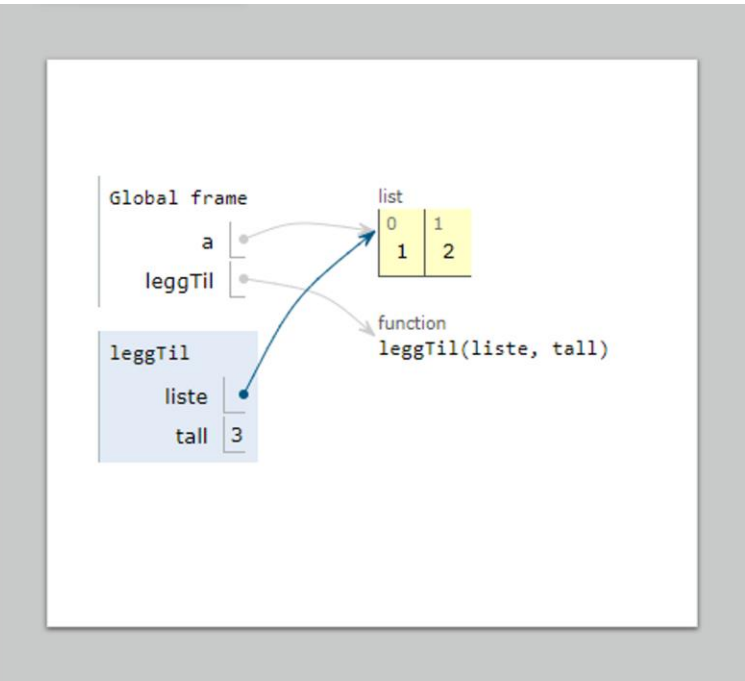
Parameteroverføring

```
a = [1,2]

def leggTil(liste, tall):
    liste.append(tall)

leggTil(a,3)
print (a)
```



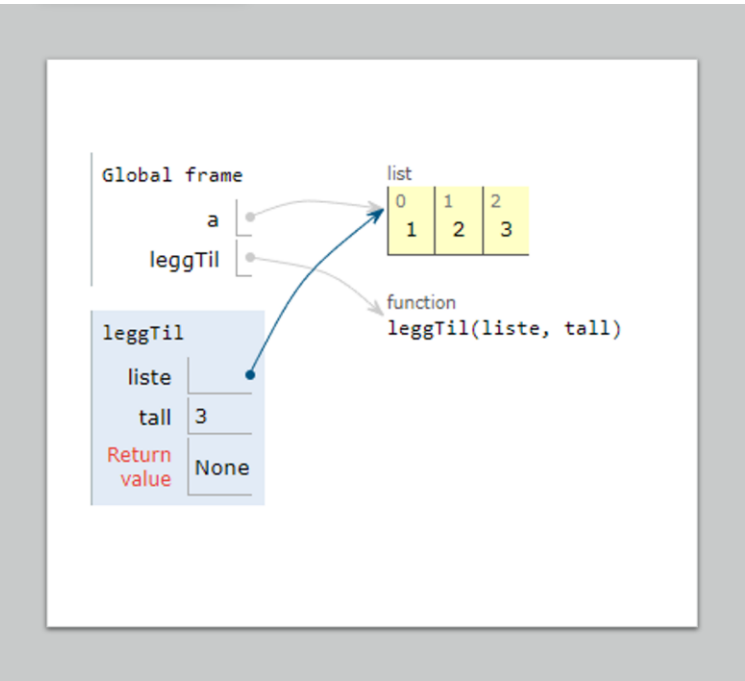
Parameter - overføring

```
a = [1,2]

def leggTil(liste, tall):
    liste.append(tall)

leggTil(a,3)
print (a)
```



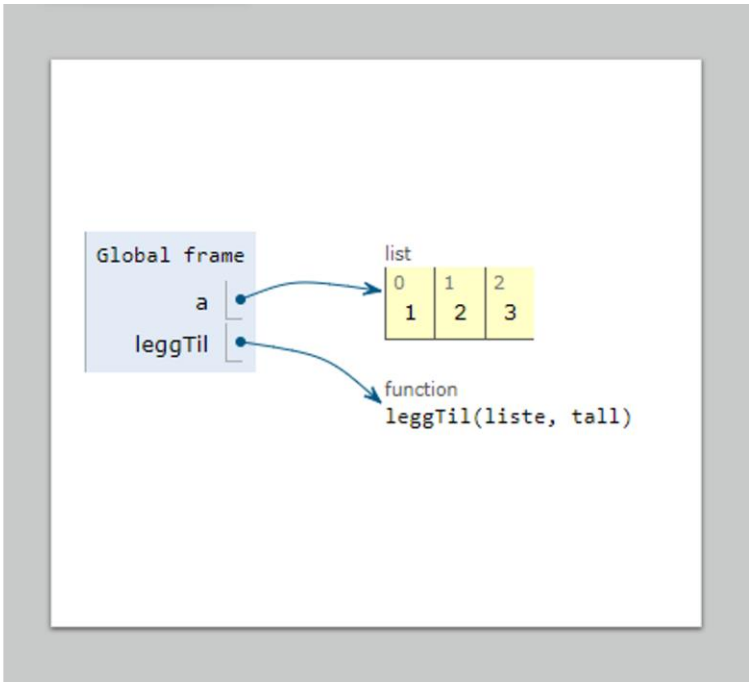
Parameter - overføring

```
a = [1,2]

def leggTil(liste, tall):
    liste.append(tall)

leggTil(a,3)
print (a)
```



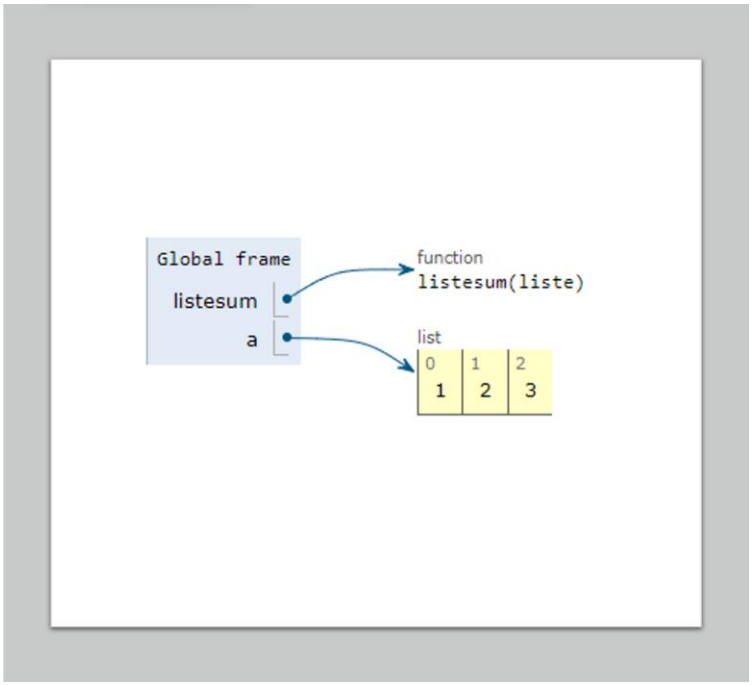
Parameter - overføring

```
a = [1,2]

def leggTil(liste, tall):
    liste.append(tall)

leggTil(a,3)
print (a)
```



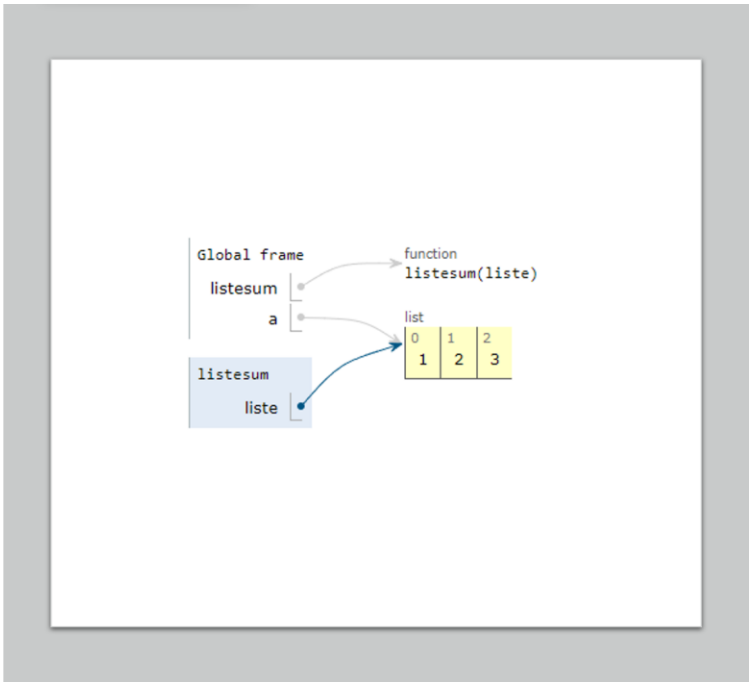
Med mye distraksjoner

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum

a = [1,2,3]
print(listesum(a))
```



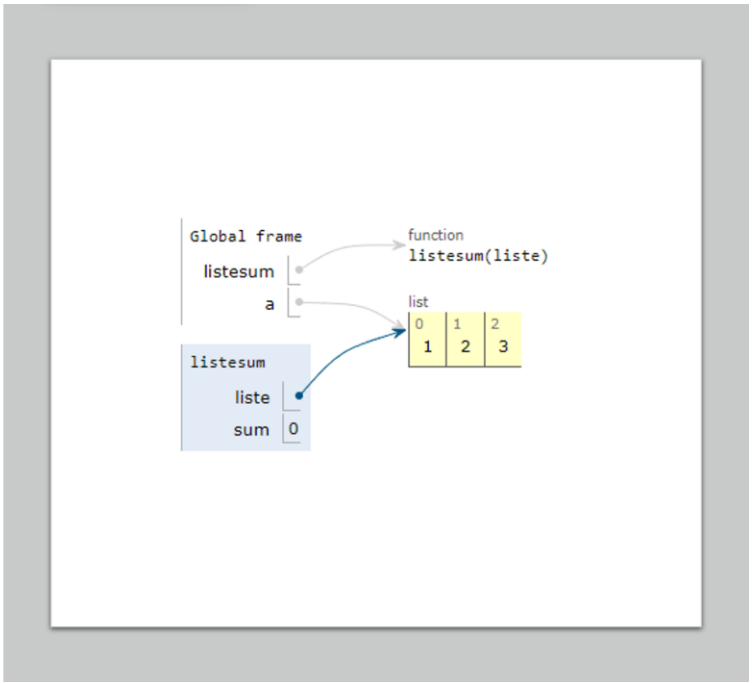
Med mye distraksjoner

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum

a = [1,2,3]
print(listesum(a))
```

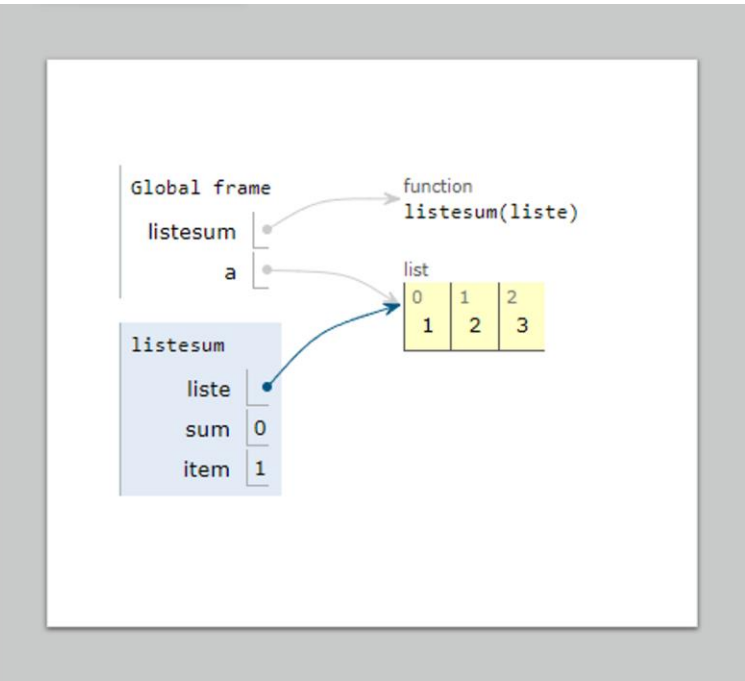
Med mye distraksjoner

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum

a = [1,2,3]
print(listesum(a))
```

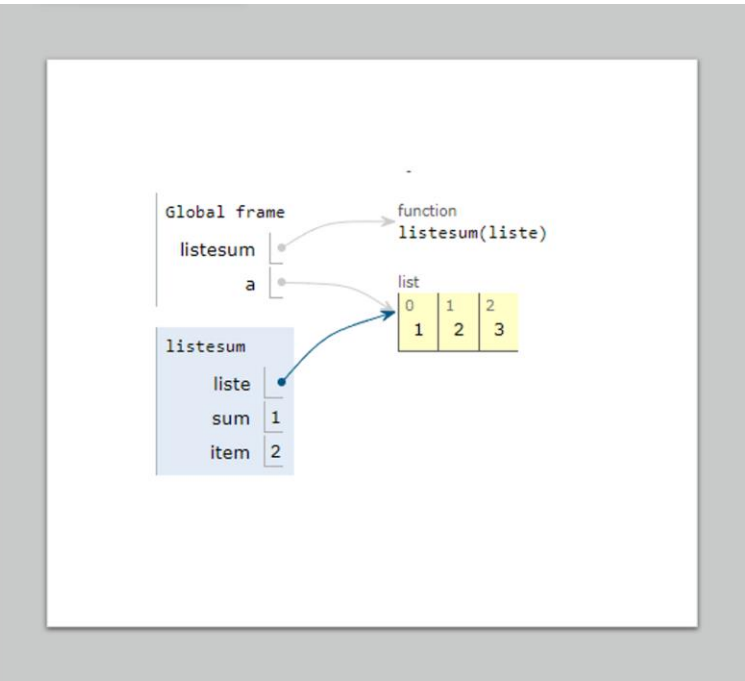
Med mye distraksjoner

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum

a = [1,2,3]
print(listesum(a))
```



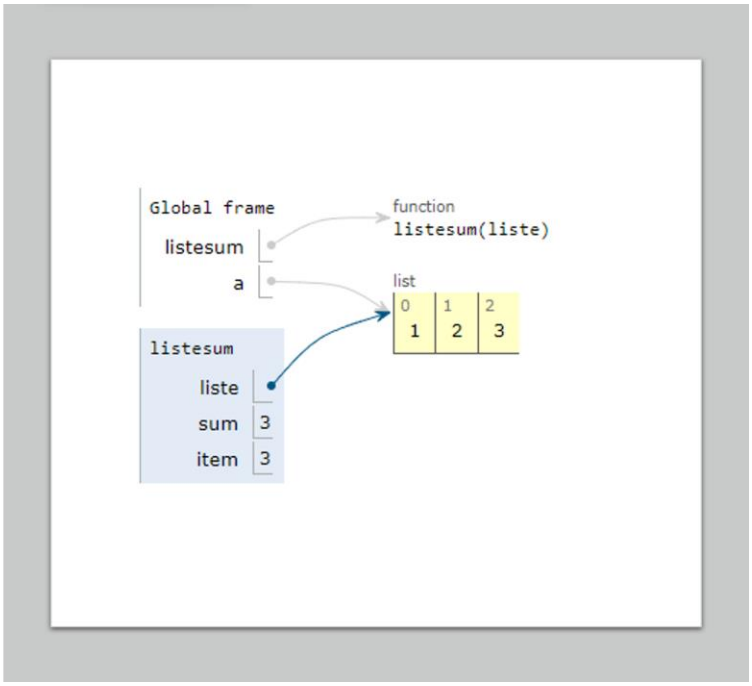
Med mye distraksjoner

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum

a = [1,2,3]
print(listesum(a))
```



Med mye distraksjoner

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum

a = [1,2,3]
print(listesum(a))
```



Med mye distraksjoner

```
def listesum(liste):
    sum = 0
    for item in liste:
        sum += item
    return sum

a = [1,2,3]
print(listesum(a))
```