Students' mental models of references in Python*

Kristin Marie Rørnes Ragnhild Kobro Runde
Siri Moe Jensen

Department of Informatics
University of Oslo

Abstract

This paper reports on a study exploring students' understanding of references and reference assignments. Students in an introductory programming course in Python were interviewed with respect to what happens during execution of reference assignment statements and function calls involving references. Previous research on Java has identified two types of mental models related to reference assignment, which in this paper is referred to as the "Copy value model" and the "Copy reference model".

An important result in this paper is that each of these models can be divided into two sub-models, giving a total of four different models where only one of them is valid. In addition, we have identified three types of mental models related to references in function calls. This gives valuable insight into students' thinking, which can then be addressed by teachers both in class and in formative assessments.

Furthermore, students in two introductory courses were asked to participate in a survey with multiple choice questions asking the students to identify the correct results of executing code examples involving references. The patterns of answers were analyzed based on the mental models identified in interviews. It was found that the identified mental models explained the most common patterns in the student responses.

1 Introduction

In standard introductory programming courses, variables and assignments are among the first programming concepts encountered by the students. A number of studies have identified typical mistakes made by students regarding what actually happens even in the simplest examples of variable assignments, and that some of these persist well into the first or even second year of programming

*This paper is based on the master thesis of Kristin Marie Rørnes: "Mental Models in Programming: Students' understanding of references in Python" [1].

This paper was presented at the NIK-2019 conference; see http://www.nik.no/.

education [2, 3, 4]. When references are introduced, students seem to find both reference types and reference assignments an even more difficult concept [5, 6, 7].

In [5, 8], Ma et al established the importance of valid mental models of references, and showed that as much as 83% of the students participating in the study held invalid mental models of reference assignment. A mental model is *an explanation of someone's thought process about how something works in the real world* [9], and [5] defines a mental model as valid when the following two conditions are met:

1. The mental model has to match with the model of how a programming concept actually works (appropriate).

2. The mental model "always" has to match with the actual model (consistent).

[5] identified different invalid mental models of references and reference assignments in Java. With an increasing number of introductory programming courses now using Python [10], it is important to establish whether or not the mental models held by the students are the same in Python as was found for Java. Building on the work in [5], this paper presents mental models of references in Python found among students in IN1000[1], the main introductory programming course at the Department of Informatics, University of Oslo. Results from a survey among IN1000-students are compared with results from a similar survey distributed among students in BIOS1100[2].

By establishing why students make certain errors, it is possible to create lectures and exercises that purposefully provoke certain ways of thinking. [4] recommends that research on computer science education should focus more on how students form their understanding of programming concepts, instead of documenting their misconceptions. They suggest that researchers should focus on how students are changing their mental models, in order to develop better learning tools and teaching strategies.

The rest of this paper is structured as follows: An introduction to references in Python is found in Section 2. In Section 3 we present the method used in this survey, before presenting the results in Section 4. Finally, in Section 5 we compare our results with those of Ma et al in [5], and discuss their implications for computer science teachers and further research in this area.

2 References in Python

This paper is concerned with the mental models the students create of the programming concept *references*, and not the details of programming specifics. It is, however, necessary to understand how a reference behaves in order to determine what a valid mental model of references is in Python.

References and reference assignments are treated slightly different in different programming languages. In Python, which is the scope of this paper, the operator = is used for reference assignments, wheres names (or reference variables) are

[1] Introduction to Object-oriented Programming, `https://www.uio.no/studier/emner/matnat/ifi/IN1000/`

[2] Introduction to Computational Modelling in the Biosciences, `https://www.uio.no/studier/emner/matnat/ibv/BIOS1100/index.html`, a new course where students learn introductory programming in Python in order to understand more about biology.

assigned reference values. These values represent memory addresses, which denote where the objects are stored on the heap. Reference assigments can be visualised in the same way for all Python objects, regardless of whether they are mutable (may be changed, e.g. lists) or immutable (may not be changed, e.g. integers and strings). A simple example is shown in Figure 1.

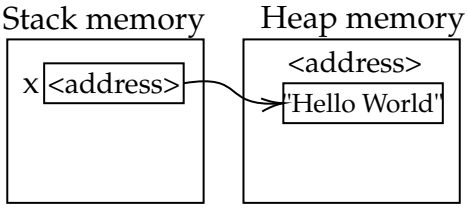


Figure 1: Visualisation of the result of the assignment `x = "Hello World"` in Python

In this paper we follow the terminology used in the syllabus of IN1000, where a reference denotes the memory location of an object, and a variable that contains the memory location to an object is *referring* to that object [11].

3 Method

The study reported in this paper was conducted at the Unversity of Oslo in Autumn 2018. Following the advice of Goldmann et al in [6], interviews were used to get a better insight into students' mental models of references. Nine IN1000-students were recruited to participate in individual interviews. The interviews were semi-structured and centered around code snippets with 4–6 lines of Python code. The participants were asked to think aloud and explain how the code would be executed. Throughout the interviews, the participants were able to speak freely and ask questions along the way. The interviewer guided them through the tasks and continued to ask questions about their thought process. Occasionally they were asked to draw what they thought would happen and they mostly used the conceptual model with boxes and arrows. At times, the interviewer also used the box-and-arrow model to make sure the participants' explanation was understood.

To supplement the data from the interviews, we also distributed an online survey among students in IN1000 and BIOS1100. The questionnaire consisted of ten multiple-choice questions partly inspired by those of [5]. The form was shared in the lab sessions to increase the possibility that students would answer and decrease the risk of students answering multiple times. This meant, however, that not all students would have a chance to answer the questionnaire. In total, we received answers from 76 students in IN1000 and 70 students in BIOS1100.[3] At the time of the survey, the students had learnt about the most basic concepts in progamming, like different data types (including lists), loops, and functions, but not classes and objects.

An important difference between IN1000 and BIOS1100, resulting in two different questionnaires being used for the two student groups, is that BIOS1100

[3]In comparison, 134 students were qualified for final exam in BIOS1100 fall 2018 (students retaking the exam omitted). The corresponding number for IN1000 is 472.

Listing 1: Hei verden

```
1  a = "Hello World"
2  b = "Hei Verden"
3  a = b
4  print(a)
```

teach that functions must have an explicit return statement, whereas IN1000 distinguishes between functions, which have explicit return statements, and procedures, which do not have explicit return statement.

Both questionnaires contained the code in Listing 1 as a control question. Students answering this question wrong are likely to have an incorrect understanding of basic assignments and the equal sign (=), which would influence their answers also to the remaining questions regarding references. 5 of 76 students from IN1000 and 5 of 70 students from BIOS1100 answered this control question wrong and were removed from the data set.

The questionnaires were anonymous. Each question contained 4–6 lines of Python code and asked what would be printed after the code was executed. The students had the possibility of choosing one of three given alternatives or writing a free-text answer if they thought the code would result in another answer than the alternatives. The questions were given in a random order to prevent students from collaborating. The alternatives, however, were not given in a random order but followed the same structure for each question to minimise any confusion. All of the code snippets and alternatives used in the questionnaires may be found in [1].

4 Results

In this section, we first present the identified mental models for references and reference assignment not involving functions, and then the identified mental models for references combined with functions. The mental models are then applied to the questions from the two questionnaires, and compared with the students' answers to see if the identified mental models could explain the patterns seen in the students' answers.

Mental models of references

The mental models presented here are based on explanations, drawings and patterns found in the transcribed interviews. There are two types of models that were clearly seen in the interviews:

Copy reference models: An understanding where the students explain `a=b` as meaning that `a` and `b` are referring to the same memory location (or "pointing" to the same object), which means that these variables have the same value (which is a reference). There are two variants of copy reference models, one invalid and one valid as will be demonstrated below.

Copy value models: A (mis-)understanding where the students explain `a=b` as meaning that `a` becomes a copy of the value that `b` has been assigned to.

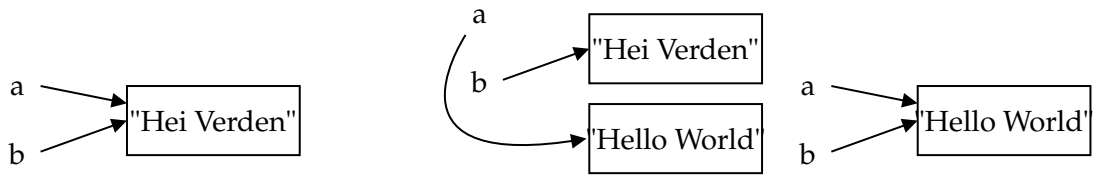
The copy value models are considered invalid, as there are cases where they give an incorrect result.

To illustrate the different mental models as found in the interviews, the code snippet in Listing 2 will be used.

Listing 2: Hei verden

```
1  a = "Hei Verden"
2  b = a
3  a = "Hello World"
4  print(b)
```

Figure 2 visualizes two different "Copy reference models" applied to the code in Listing 2. In both models, the assignment `a=b` results in `a` and `b` referring to the same text object as seen in Fig 2a. The difference between the two models can be seen in Figure 2b and Figure 2c. In the valid model MREF1, `a` is *reassigned* to a new memory location after code line 3 is executed, but the object that `b` refers to is not changed. In the invalid model MREF2, the value in `a` is changed *at the memory location* that `a` refers to after code line 3 has been executed, which means that the object `b` refers to also has changed. MREF2 may seem illogical, in that the assignment statement is interpreted differently depending on whether the right side is another variable (in line 2) or a value. However, many of the students seem to use this line of reasoning in a consistent manner.



(a) MREF1 and MREF2 after line 2 is executed

(b) MREF1 after line 3 is executed

(c) MREF2 after line 3 is executed

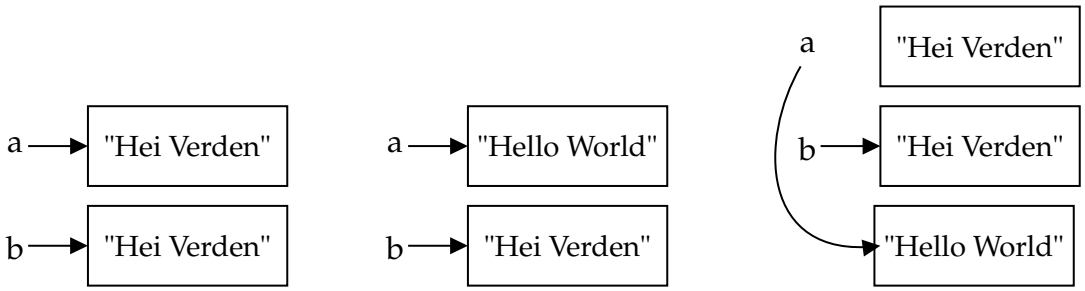
Figure 2: MREF1 and MREF2 for Listing 2

Figure 3 visualizes two possible "Copy value models". Although in this case, both result in the text `"Hei Verden"` being printed in line 4 (as in the valid model MREF1), these models are not considered valid since there are other cases where the result is not correct, as will soon be illustrated.

As can be seen in Figure 3a, the copy value models differ from the copy reference models in that here, `a` and `b` are understood to refer to (or point to) two different objects that contain the string object `"Hei Verden"`.

In Figure 3b the version called MVAL1 is presented. Here, the value in `a` is changed *at the memory location* that `a` refers to after code line 3 has been executed. In Figure 3c the version called MVAL2 is presented. In this version, `a` is *reassigned* to a new memory location after code line 3 is executed. This is a subtle difference, but important e.g. to distinguish MVAL2 from the valid model MREF1, which both explains code line 3 as a reassigment of `a`.

A case where MVAL1 and MVAL2 both give an incorrect result, is provided in Listing 3. Here, the correct result of the print statement in line 4 is `[10,2,3]`, while



(a) MVAL1 and MVAL2 after line 2 is executed

(b) MVAL1 after line 3 is executed

(c) MVAL2 after line 3 is executed

Figure 3: MVAL1 and MVAL2 for Listing 2

Listing 3: studentID

```
1 personID = [1,2,3]
2 studentID = personID
3 personID[0] = 10
4 print(studentID)
```

students using MVAL1 or MVAL2 would say that the result is `[1,2,3]` since they believe that `studentID` is a *copy* of `personID`, meaning that changes to `personID` are not reflected in `studentID`.

The questionnaires for IN1000 and BIOS1100 consisted of respectively four and five simple code snippets where the four models identified here could be applied directly. As can be seen from Table 1, only 38 of the 136 students answered consistently in accordance with the valid model MREF1. For almost 70% (68 of 90) of the remaning students, their answer patterns could be explained using one of the identified invalid models. However, the surveys did not include any questions to distinguish between the invalid models MVAL1 and MVAL2, as the difference between these had not been identified at the time of the surveys.

Model	**Notes**	**IN1000 (N=71)**	**BIOS1100 (N=65)**	**Total (N=136)**
MREF1	Copy reference (valid)	17	21	38
MREF2	Copy reference (invalid)	9	11	20
MVAL1/2	Copy value (invalid)	34	14	48
(undefined)	(invalid)	11	19	30

Table 1: Correlation between mental models and students' answers on questions without functions

Mental models of references combined with function calls

Introducing functions with references as parameters, adds to the complexity as students now also have to understand how parameter passing works, and if and when changes to the parameter inside the function body affects the rest of the program. Although having worked with functions and parameters in the course, none of the interviewed students could correctly explain what happens when a

function is called. Instead, most of their explanations were found to fall into one of two categories: 1) A copy of the argument's value is sent to and stored in the parameter of the function (the model FCOP), and 2) The argument is the actual object which is sent to the function as a parameter (the model FOBJ). To illustrate the different models, we will use the code example given in Listing 4.

Listing 4: language

```
1 language = ["English","Norwegian","German"]
2 programmingLanguage = ["Python","Java","Simula"]
3 def changeLanguage(array1,array2):
4   array1 = array2
5 changeLanguage(language,programmingLanguage)
6 print(language)
```

Before presenting the invalid models found in the interviews, we present a valid mental model (the model FREF) based on how this is taught in IN1000 and how Python actually works. When a function with parameters is called, local variables are created with the parameter names. These variables become references to the same objects as the arguments pased to the function are referring to. In Figure 4, the function call to `changeLanguage` in line 5 means that the parameters `array1` and `array2` become local variables that refer to the same objects as the arguments `language` and `programmingLanguage` refer to in the global scope.

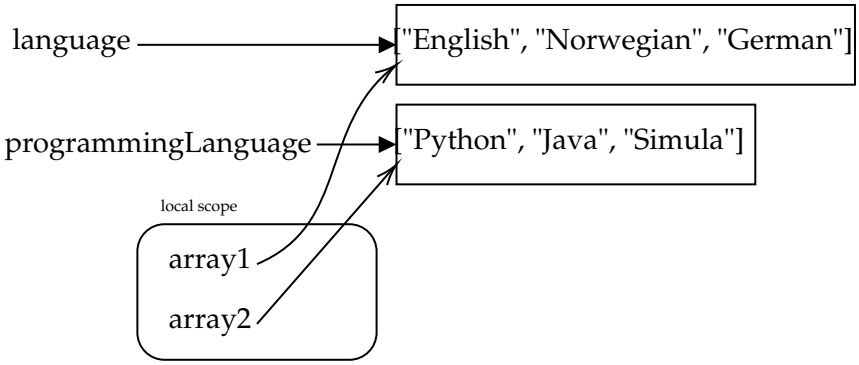


Figure 4: FREF after line 5 (i.e. before execution of the function body in line 4) in Listing 4

For the function body in line 4 in Listing 4, the combination of FREF with the valid model MREF1 for reference assignment is illustrated in Figure 5, which shows that the result of the `print`-statement in line 6 will be the list `["English","Norwegian","German"]`.

As explained above, some students have an invalid model FCOP where the parameters of a function becomes *copies* of the values referenced by the passed arguments, as illustrated in Figure 6. The parameters are treated as local variables and any changes inside the function will only affect the local copies. Combining FCOP with one of the four models for reference assignment (MREF1/2 and MVAL1/2), would result in a different understanding of what happens inside the function body, but as this would not have any effect outside the function, the effect of the print statement would be the same in all cases.

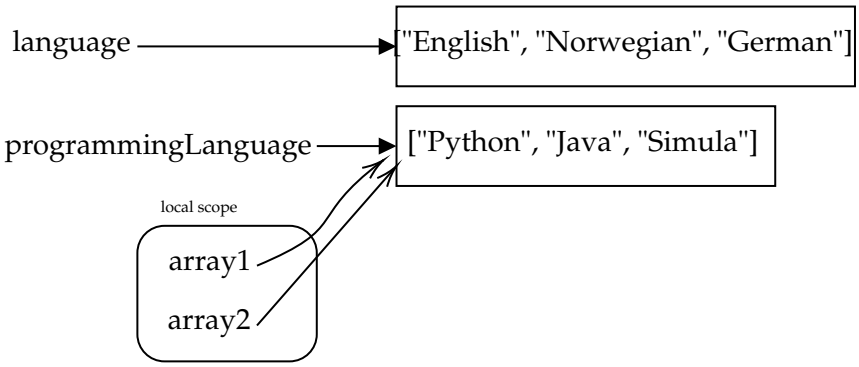


Figure 5: FREF+MREF1 at the end of line 4 in Listing 4

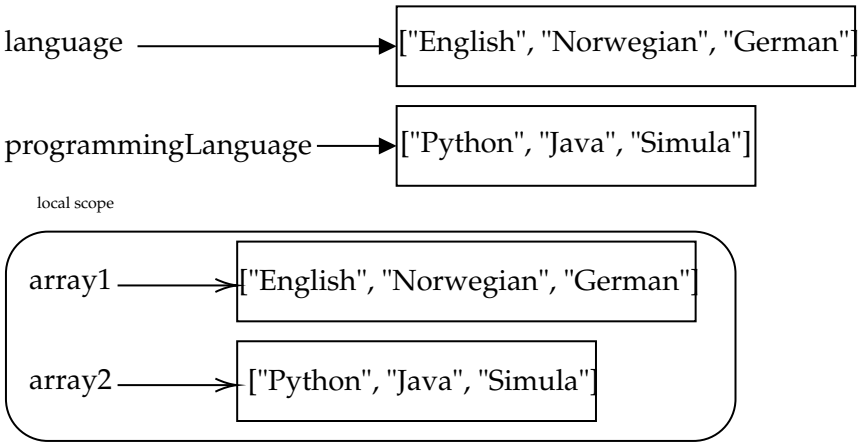


Figure 6: FCOP after line 5 in Listing 4

For references combined with functions, the other invalid model identified in the interviews is the model FOBJ where the parameters are believed to be set to the *actual* objects passed to the function. Figure 7 show that in this case, `array1` is identified with `language` and `array2` is identified with `programmingLanguage`. All the changes made to the local variables inside the function will affect the references and objects in the global scope.

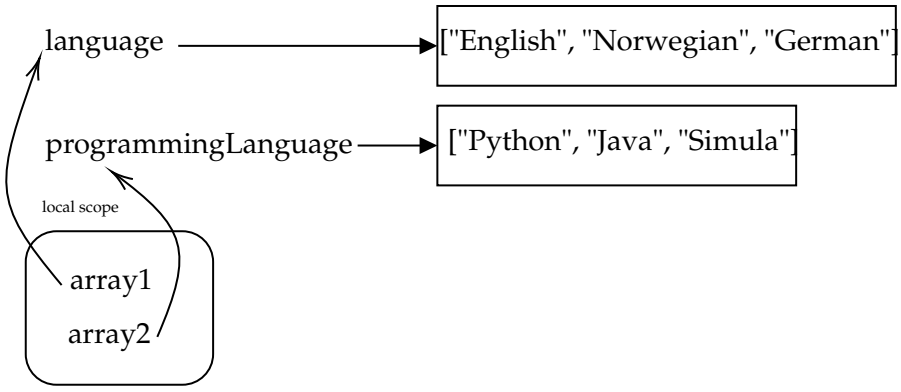


Figure 7: FOBJ after line 5 in Listing 4

Combining the invalid model FOBJ with the invalid copy value models MVAL1/2 results in an incorrect result as illustrated in Figure 8. As seen, the value of the object referred to by `programmingLanguage` is copied to the object

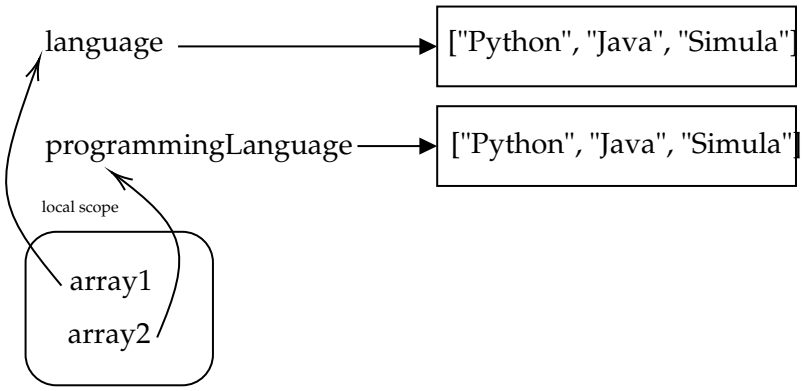


Figure 8: FOBJ+MVAL1/2 at the end of line 4 in Listing 4

that `language` is referring to.

The combination of FOBJ with the copy reference models MREF1 (valid) or MREF2 (invalid) is illustrated in Figure 9. As shown, `language` is now referring to the *same* object as `programmeringLanguage` is assigned to. As FOBJ is an invalid model, combining it with MREF1 will often give an incorrect result (as in this case), even though MREF1 in isolation is a valid model.

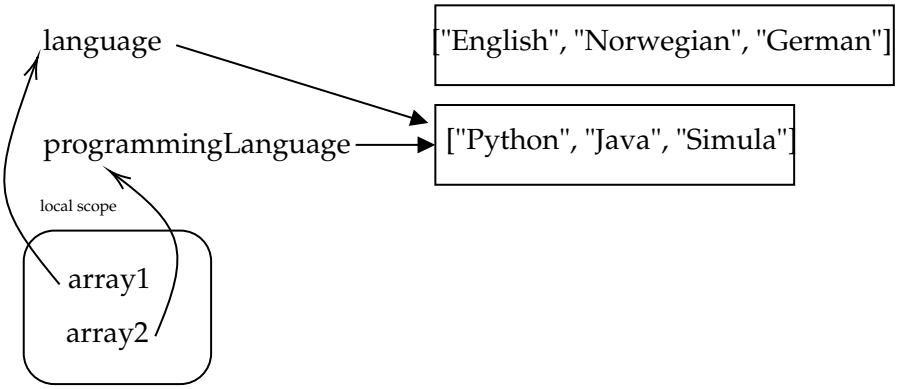


Figure 9: FOBJ+MREF1/2 at the end of line 4 in Listing 4

The questionnaires for IN1000 and BIOS1100 consisted of respecively two and four simple code snippets relevant for the identified mental model of references combined with functions.[4] As can be seen from Table 2, as little as 14% (19 of 136) answered in accordance with the valid model FREF, meaning that the remaining 86% either have an invalid/inconsistent model or have restorted to guessing. Most of the answers from the IN1000 students could be explained by one of the identified models, but only less than half of the answers from the BIOS1000 students.

Finally, we have compared the students' answers to the questions with and without functions and how they fit with the identified models. The results for IN1000 is presented in Table 3.[5] As can be seen, only 4 students (6%) have

[4]The questionnaire for IN1000 also included four other questions concerning functions, but during the analysis it was found that wrong answers here were most likely due to misunderstanding other Python specific concepts and not references as such. They are therefore not included in the results presented here.

[5]The results for BIOS1100 may be found in [1].

Model	Notes	IN1000 (N=71)	BIOS1100 (N=65)	Total (N=136)
FREF	valid	8	11	19
FCOP	invalid	22	7	29
FOBJ	invalid	34	10	44
(undefined)	(invalid)	7	37	44

Table 2: Correlation between mental models and students' answers on questions with functions

results consistent with the valid models MREF1 and FREF. For the remaining students potententially using the valid model MREF1, the invalid model FOBJ is the most common, indicating that while they correctly understand that reference assignment means reassigning the variable, they incorrectly believe that reference parameters is identified with the actual parameter names, and not their content.

For the invalid models MREF2 and MVAL1/2, most students seem to also use one of the invalid models FCOP and FOBJ, with the students almost evenly split between the two.

	FREF	FCOP	FOBJ	Other
MREF1	4	2	10	1
MREF2	0	3	5	1
MVAL1/2	2	14	14	4
Other	2	3	5	1

Table 3: Comparing the results from IN1000-students on questions with and without functions (N=71)

5 Conclusions and future work

Based on the results from student interviews, we have in this paper presented four mental models for reference assignment, and three mental models for functions with references as parameters. Student questionnaires from two introductory courses in programming show that very few students have valid mental models of references more than halfway through the course, and there are strong indications that the most common patterns in the student answers may be the result of using one of the identified models.

For reference assignment, the "Copy reference models" is similar to the "Component assignment model" in [5] and the "copy value models" is similar to the "Set equal model" in [5]. In this paper, we have explained these two types of models in more detail, and found that each type may be divided into two sub-models, all found among the students interviewed as part of this study.

Assuming that the students did not answer the questionnaires simply by guessing or answering too quickly, the results show that some students have answer patterns not consistent with any of the identified models. As this applies to students from BIOS1100 more than students from IN1000, this could be the result of teaching differences not captured in the interviews with only IN1000-students. A natural follow-up study would be to interview more students from

different programming courses.

At the Department of Informatics, University of Oslo, Java is used as the main programming language in the programming courses following IN1000. In principle, the models presented in this paper should be applicable to any standard object-oriented programming language. To learn more about students' understanding of references, and how this understanding develop over time and with different programming languages, it would be interesting to conduct a similar study with interviews and multiple-choice questions in one or more advanced programmering courses, and preferrably also a more longitudinal study following the same students over time.

One of the questions that emerge from these findings is how to present references to students who learn Python, and in particular how this may done in a way that is applicable or at least easily transferable to Java and other programming languages. This is ongoing work at the department, and preliminary results indicate that targeted interventions at least to some extent increase the number of students with a valid mental model of references and reference assignment.

We believe that being aware of how difficult the concept of references is for students, and the existence of the different invalid mental models, may help teachers pay more attention to this both in lectures and when helping individual students. As part of future research, a set of test questions should be developed to be able to clearly distinguish between the different models identified in our study. A precise set of questions will also be useful as a diagnostic tool for teachers and/or the students themselves. As we have seen, invalid mental models may give a correct answer in many cases, which may make it more difficult for the students to realize that a particular programming error may be due to a lack of understanding of references.

References

[1] K. M. Rørnes. Mental models in programming: Students' understanding of references in python. Master's thesis, Department of Informatics, University of Oslo, Norway, 2019.

[2] B. D. Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[3] Simon. Assignment and sequence: Why some students can't recognise a simple swap. In *Proc. of the 11th Koli Calling International Conference on Computing Education Research*, Koli Calling '11, pages 10–15. ACM, 2011.

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

[5] L. Ma, J. Ferguson, M. Roper, and M. Wood. Investigating the viability of mental models held by novice programmers. In *Proc. 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 499–503. ACM, 2007.

[6] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. Identifying important and difficult concepts in introductory computing courses using a delphi process. In *Proc. 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, pages 256–260. ACM, 2008.

[7] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman. Identifying student misconceptions of programming. In *Proc. 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 107–111. ACM, 2010.

[8] L. Ma, J. Ferguson, M. Roper, and M. Wood. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, 21(1):57–80, 2011.

[9] Wikipedia contributors. Mental model — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Mental_model&oldid=914045188`, 2019. [Online; accessed 7-September-2019].

[10] R. M. Siegfried, D. Liporace, and K. G. Herbert-Berger. What can the Reid list of first programming languages teach us about teaching CS1? In *Proc. of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE 2019*, pages 1256–1257, 2019.

[11] C. S. Horstmann. *Python for everyone*. Wiley, 2nd edition, 2016.