Knowledge Transfer and False Friends: Insights on Transitioning from C to Java

Yifan Du
University of Technology Chemnitz

Belinda Schantong
University of Technology Chemnitz

Janet Siegmund
University of Technology Chemnitz

***Abstract*—Background: When acquiring new programming languages, learners typically transfer their knowledge from previous languages to the new one. Recent studies demonstrate that this transfer is similar to the process of learning a second language, so learning can be more efficient, but also interference can take place, similar to false friends. While these studies demonstrate the existence of transfer, they have focused on programming languages with rather larger differences, such as Python versus Java.**

Objective: Our goal is to understand how students transfer between two similar programming languages, that is, from C to Java, adding a different angle to how transfer takes place.

Method and Results: To this end, we gave students a Java comprehension test in the first week of a CS2 course, after they have completed a CS1 course based on C. We could confirm that students transfer knowledge from C to Java, including false friends. We repeated this study a year later, but included a dedicated Java tutorial before conducting the Java comprehension test. With these explicit instructions, students experienced less interference when learning Java, demonstrating that they profit from being made explicitly aware of the syntactical intricacies of Java. However, we also observed persistent interference for some concepts, indicating the need for more in-depth instructions.

***Index Terms*—Programming language transfer, Program comprehension, Novice programmers, Java, Syntax and semantics.**

I. Introduction

Programming learning is becoming more ubiquitous as a growing number of countries are integrating programming into their school curricula [1]–[3]. When transitioning between schools, school levels, or into higher education, students will most likely not only learn one, but several programming languages over time. Thus, transitioning between programming languages is essential.

Just as with learning natural languages, typically, learning a new programming language builds on existing knowledge of previously learned programming languages, such that knowledge from one language is transferred to the new one. And just as in natural language, this works well sometimes, for example, when there are similarities in vocabulary, such as from German `Fisch` to English `fish`, or from French `manger` and Italian `mangiare` (both mean 'to eat'). Sometimes, however, it is difficult and interference takes place, for example, in the form of 'false friends', such as English `actually` and Spanish `actualmente`, where the latter means 'at present/currently', so it is actually very different from the similar looking English word.

A language learner who is able to assume what the German word `Fisch` means based on their knowledge in English is experiencing *positive transfer*, where knowledge learned in one context has a positive effect on performance in another context. By contrast, a Spanish learner that falls for the trap of the false friend `actualmente` experiences *negative transfer*, such that the knowledge learned in one context causes negative influence on performance in other contexts. [4]

Recent studies have demonstrated how transfer takes place in learning a new programming language, which is described in the *Model of Programming Learning Transfer* [5]. According to the model, when learning a second or subsequent programming language, three different kinds of transfer can occur:

1) *Positive transfer* can occur for *True Carryover Concepts*. These are concepts that, in the new programming language, are similar in syntax and semantics compared to already known programming languages. For example, the syntax of declaring a variable of the type `int` is a concept shared by the languages C and Java with consistent underlying semantics. The expression `int number = 2;` is syntactically correct in both languages and has the same meaning.

2) *Negative transfer* is expected for *False Carryover Concepts*, which are syntactically similar in the new programming language, but not semantically. These "false friends" cause interference. For example, we can declare an integer variable and initialize it with the value 1 in Java and C. Then, we assign the value 10.5 to this variable. At this point, the syntax remains consistent in both languages, but the underlying semantics are different. In C, the number is truncated to 10 with a compiler warning. In Java, such an assignment leads to a compiler error.

3) *No transfer* occurs for *Abstract True Carryover Concepts*. These are semantically similar in the new programming language, but not syntactically. Since learners are unfamiliar with the new syntax, they cannot recognize that they already know the concept, just with a different look. One example are structures in C and classes in Java. Although C has no syntax to define classes, structures can be used to present simple classes, making the semantics similar in this context. For example, the structure `struct Point{int x; int y;}` can represent a point, similar to a `class Point {int x; int y;}` in Java.

The Model of Programming Learning Transfer has been evaluated regarding transfer from Python to Java [5], and has been successfully applied in other contexts [6]–[8]. However,

there is currently a gap on how transfer between more similar languages takes place, specifically, with comparable type systems. To address this gap, we have conducted a study to understand how students who are starting to learn Java in a CS2 course after having learned C in a CS1 course transfer their knowledge. To this end, students completed a Java comprehension test, in which they determined what a snippet of Java code would do based on their prior knowledge. To understand to what extent an explicit introduction to Java helps to support transfer and avoid interference, we have run the study twice in two separate years (2023, 2024), and have incorporated a dedicated Java tutorial in the 2024 before conducting the Java comprehension test. This is a replication of the study by Tshukudu and Cutts [5], in which we have adapted the Java questions and expected transfer from C as the previous programming language.

In a nutshell, we found evidence for all three kinds of transfer, indicating that students transfer concepts, leading to either (expected) correct responses, indicating *positive* transfer, or (expected) incorrect responses, indicating *no* and *negative* transfer. However, how students transfer their knowledge seems tightly coupled to the concrete concept (e.g., variable scope, type (in)compatibility), not whether positive or negative/no transfer is expected. To understand the effect of the concepts on how students transfer their knowledge to Java, we explored the responses of students in detail. We found that especially scoping and comparison of complex data types are difficult to get right, even after a dedicated Java tutorial. Furthermore, object assignment, string concatenation with the + operator, using attributes for arrays, and basics of classes are difficult to get right without introduction, but can profit considerably from a dedicated Java tutorial.

To summarize, we make the following contribution:

- Empirical evidence to describe transfer to a new programming language, strengthening the view that programming language learning is close to learning a second natural language.
- Details on the relationship between concrete concepts, transfer, and dedicated instructions in the context of programming learning.
- A publicly available replication package: https://github.com/CodePenguinny/Knowledge-Transfer-and-False-Friends.git

II. Related Work

Most studies on transfer between text-based programming languages have focused on transfer from Python to Java [9]–[12], but also the transfer from Matlab to C [13], from object-oriented to functional languages [14], and between different database languages [15] have been studied. One study focused on the comparison of transfer between languages that were either different in semantics but similar in syntax, *or* different in syntax but then similar in semantics [6]. Most of them fall in line with the Model of Programming Learning Transfer and support the call for explicitly leveraging the concepts of transfer in the teaching of subsequent programming languages [16]. Only the results of Lu and others [6] differ from the expectations set by the Model of Programming Learning Transfer, in that students had little trouble transferring between the syntactically different, but semantically similar languages. While all of these studies focus on learning transfer between text-based programming languages, none of the studies has focused on the transfer between syntactically similar languages with comparable type systems.

Visual programming languages are often used to ease novice programmers into programming, but transferring skills to text-based programming languages seems to be difficult [17]–[21]. However, addressing transfer explicitly might help mitigate common misconceptions from learners [7], [22]. Other dedicated approaches exist to foster transfer, such as *frame-based editing* [23] or *facets of meaning* [24], showing a visual programming language and a text-based programming language side by side [25], [26], while focusing explicitly on the underlying concepts and computational thinking [25], or using additional tools and gamification [27], [28], or bridging languages [29]. These studies investigate the transfer of knowledge in programming language learning and propose methods to alleviate the difficulties of students when transitioning from visual to text-based programming languages. Their goal is to improve teaching methods by leveraging learning transfer, similar to that of our research.

Transfer has been studied in other aspects of programming learning, such as transferring knowledge gained from training tasks to later assessment [30], which is difficult for most learners, even for very basic problems [31] and across all aspects of transfer [32]. Approaches to alleviate these difficulties exist, such as using explicit instructions and bridging techniques [33]. For example, the programming language Hedy [34] uses these techniques to leverage transfer from natural language learning to programming language learning. For transfer between programming languages, approaches are also currently being developed [7], [10], but concrete studies to evaluate their effects are sparse. Our study will contribute empirical data for fostering transfer between programming languages.

Transfer has also been observed in programming experts. Typically, they struggle less than novices with questions of syntax and semantics [35]. However, existing tutorials for experts often do not leverage prior knowledge from experts [36], and dedicated tools to avoid negative transfer are missing [37]. Studies have also stressed the importance of not only understanding differences in concepts and syntax, but also in idioms and conventions of different languages [38].

III. Experimental Design

In this section, we describe the experimental design of the study including the research questions, materials, participants, and the experiment procedure. All material is available on the project's Web site: https://github.com/CodePenguinny/Knowledge-Transfer-and-False-Friends.git.

TABLE I
QUESTIONS AND EXPECTED ANSWERS FROM THE JAVA COMPREHENSION TEST. THE EXPECTED TYPE OF TRANSFER IS INDICATED BY THE COLOR. POSITIVE TRANSFER IS IN TEAL, NEGATIVE TRANSFER IN RED AND NO TRANSFER IN BLUE

Question	Description	Core Concept	Correct Answer	Expectation C	Expectation Python
Q1	Assigns the float number "10.5" to an int variable with initial value 1.	Type compatibility	Error	`10 / 11`	`10.5`
Q2	Declares a variable "Hello" inside a loop, prints it twice, then accesses it outside the loop.	Variable scope	Error	Error	`Hello` `Hello` `Hello`
Q3	A while loop prints the values of i from 0 to 1 before the loop terminates when i reaches 2.	While-loop	`0` `1`	`0` `1`	`0` `1`
Q4	Two objects of type String (2023)/Array(2024) with the same value are created with the `new` keyword, then compared with ==, which checks for the reference equality.	Object comparison	`false`	`True`	`True`
Q5	A method that multiplies integers is called in the main method.	Methods	`14` (2023) `21` (2024)	`14` (2023) `21` (2024)	`14` (2023) `21` (2024)

Q6	Two objects (`n1`, `n2`) are created with attribute values ("51"). The attribute of n1 is printed; then, n1 is assigned to n2 (`n2=n1`), so both refer to the same object. The attribute value of `n1` is changed to "53" after which the attribute value of `n2` is printed.	Object assignment	`51`
			`53`
			`53`
				`51`
				`53`
				`51`
					`51`
					`53`
					`53`

Q7 (2023 only)	A method uses nested for-loops to find two numbers in an array whose sum equals a specified target value, and an array with the indices of the numbers is returned, using the keyword "new" to allocate memory space for the array.	Declaring arrays	`4` `5`	Error / Confusion	Error / Confusion
Q8	String concatenation using "+": Two strings ("hello" + "there"), and a string and a number ("exec" + 3) are concatenated and printed.	String concatenation	`hellothere` `exec3`	Error	`hellothere` Error

Q9	Initializes a two-dimensional array and prints its contents with nested loops, using ".length" to access the length of the array.	Array length	`1 2 3`
			`4 5 6`
			`7 8 9`
				Error	Error / Confusion

Q10	A class "Student_Info" represents a student with three private attributes: num, name, and age. It provides getter methods for all and a setter method to modify age. In the main method, an object is created with the values "001", "Max", and "25". These values get printed. Then, the age gets updated to "23" and printed.	Classes	`001,Max,25` `23`	Confusion	`001,Max,25` `23`

A. Research Questions

Our study investigated how novices transfer their knowledge of previous programming languages to learning Java in a CS2 course. Our focus is how students who learned C in a CS1 course transition to Java in a subsequent CS2 course. Both programming languages have a static type system (different from Python), so possibly we can observe positive transfer for concepts related to types. Furthermore, the sample in our course is diverse regarding previous experience. Most students have learned C, but we cannot exclude that they have also learned other programming languages. It is quite common that students are familiar with both, C and Python, and few students joined our CS2 course from a different course that did not teach C, but Python or other programming languages. Thus, we can also observe transfer from other programming languages to Java. We phrase the first group of research questions:

RQ1: What transfer can we observe when students transition to Java...

- RQ1.1: ...from C?
- RQ1.2: ...from other programming languages?

To answer these research questions, we repeat the study of Tshukudu and Cutts [5] in our courses.

Having understood how students transition to Java, we can integrate dedicated training to support students in transferring their knowledge to Java. This way, we can improve current teaching approaches and exploit students' previous knowledge, easing their way into Java. We define the second group of research questions:

RQ2: What is the effect of explicit Java grammar instruction on transfer ...

- RQ2.1: ... from C to Java?
- RQ2.2: ... from other programming languages to Java?

To answer our second group of research questions, we conduct the study two times in our courses, first in 2023,

and second, in 2024, in which we start with a dedicated Java tutorial.

B. Material

For our study, we adapted the code comprehension snippets from the study by Tshukudu [10]. They contained basic Java concepts, such as declaration of variables, declaration of objects, usage of methods, and so on. An overview of the questions and which concepts they contain can be found in Table I. The full code snippets can be found on the project's Web site.

To observe transfer, each snippet was designed to reflect one of the three transfer types: positive transfer, negative transfer, and no transfer. Thus, each snippet contains a programming concept that is expected to lead to exactly one type of transfer, which depends also on the previously known programming language(s) of students.

```
for (int i=0; i<2; i++) {
    String var="Hello";
    System.out.println(var);
}
System.out.println(var);
```

Listing 1. Java Code Snippet Example

As an example of positive (from C) and negative (from Python) transfer, we show the code of Question 2 in Listing 1. This code will lead to an error, because the variable `var` is accessed outside the loop in which it is defined, violating the variable's scope. Since C has similar scoping rules as Java, we assume that students coming from C correctly respond that this code produces an error. By contrast, Python has less restricted rules of scoping, so accessing the variable outside the loop is syntactically correct Python code. Thus, we expect that, when students transfer their Python knowledge to this Java code, they respond with three times "Hello".

As an example for no transfer (from C), Question 10 includes the definition of a small class and the use of the keyword `new`. Since there is no syntactic similarity in C, we expect that students cannot transfer their knowledge from C, even though there are comparable concepts with the definition of `struct` and the function `malloc`.

The task for the participants was to comprehend Java snippets and determine what the output of the code would be, either with no introduction to Java (2023) or a dedicated (2024) tutorial to Java. This way, we ensure that participants needed to base their answers on their prior programming knowledge (2023). The snippets could also lead to errors, which we told participants before they started the Java comprehension test, and instructed them to shortly describe an error if they thought there was one.

In the 2024 instance, we removed Q7 after observing that it tended to cause confusion. We did not replace it, because many participants lost interest after completing six tasks. We also made some slight adjustments to Q4, Q5 and Q10 to reduce confusion that was caused by details in the snippets, not intricacies of the concepts, such as unused parameters or the format of the output.

To assess the background of participants, we used an established questionnaire to measure programming experience, which is based on self estimation, and also asked about their familiarity with different programming languages [39].

C. Participants

Most participants are students who completed a CS1 course in the preceding semester. We present an overview of our sample in Table II. In total, 104 students participated in the experiment in 2023. Before data analysis, we removed 13 invalid responses and 7 students who had previous experience in Java, leaving 84 participants. Most students had learned C in a preceding CS1 course, and some had prior experience with Python and/or other languages.

In the 2024 instance, 69 students participated, of which we removed 6 invalid responses and 13 students with previous Java experience, leaving 50 participants.

D. Conduct

The experience questionnaire and comprehension test were implemented in LimeSurvey and distributed by giving the students the link to the online survey. We also provided a paper-based version to students who preferred it.

In 2023, the survey was conducted during the first lecture of the semester. In 2024, the survey was conducted in the *second* week of the semester, after having received a dedicated Java tutorial. The tutorial provided an overview of the fundamental components of Java, which we noticed were especially difficult in the 2023 instance. It included control structures, primitive and complex data types, and the operations that can be executed on them. Additionally, the tutorial covered the use of one- and two-dimensional arrays and introduced the basics of classes. We used live coding, during which students could ask questions to explore Java constructs.

In 2023, the order of the questions was fixed to ensure an even distribution of questions containing different types of concepts. However, as participants grew fatigued, the number of responses decreased considerably for the later tasks. To have a more evenly distributed number of answers per question, we pseudo-randomized the order in 2024.

IV. Results

A. Data Preprocessing

Before analyzing the data, we removed invalid answers which include blank answers and duplicated submissions: Some students gave up answering the Java comprehension test

TABLE II
Programming Experience Distribution of Participants

Year	C	Python	C & Python	None	Other	Sum
2023	45	4	19	7	9	84
2024	16	2	15	13	4	50

TABLE III
PERCENTAGE OF CORRECT ANSWERS PER QUESTION AND KNOWLEDGE BACKGROUND IN 2023 AND 2024. POSITIVE TRANSFER IS IN TEAL, NEGATIVE TRANSFER IN RED, AND NO TRANSFER IN BLUE.

2023	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Average	14.61%	14.61%	67.42%	5.68%	78.82%	15.38%	3.28%	39.39%	31.91%	38.64%
C	11.63%	16.28%	62.79%	4.65%	74.42%	7.14%	0.00%	36.11%	20.83%	29.17%
Python	25.00%	0.00%	75.00%	0.00%	100.00%	0.00%	0.00%	50.00%	50.00%	-
C & Python	15.79%	15.79%	78.95%	5.26%	77.78%	33.33%	7.69%	46.15%	40.00%	44.44%
None	14.29%	0.00%	57.14%	0.00%	57.14%	14.29%	0.00%	14.29%	0.00%	28.57%
Other	0.00%	0.00%	44.44%	0.00%	77.78%	0.00%	0.00%	11.11%	22.22%	11.11%
Valid Answers	82	82	82	81	78	71	56	60	43	40
2024										
Average	25.81%	21.67%	80.00%	15.25%	91.80%	32.76%		52.46%	69.49%	71.19%
C	18.75%	14.29%	93.33%	6.67%	93.75%	26.67%		50.00%	80.00%	68.75%
Python	0.00%	0.00%	50.00%	0.00%	100.00%	0.00%		100.00%	0.00%	0.00%
C & Python	40.00%	33.33%	78.57%	7.69%	85.71%	35.71%		42.86%	71.43%	78.57%
None	7.69%	15.38%	61.54%	15.38%	100.00%	7.69%		61.54%	53.85%	53.85%
Other	50.00%	25.00%	75.00%	25.00%	75.00%	50.00%		50.00%	50.00%	75.00%
Valid Answers	50	48	48	46	48	47		48	47	47

after the experience questions, while others skipped certain questions. A few students submitted their responses twice, in which case we kept the final submission.

B. RQ1: What transfer can we observe when students transition to Java?

To answer RQ1, we evaluated the correctness of answers, illustrated in Table III for both instances.

1) RQ1.1: Transfer from C to Java: For the questions in which we expect *positive* transfer, we see comparable results in both instances: Q3 and Q5 have high correctness rates, especially in the 2024 instance. Q2 has a consistently low correctness rates in both instances. Thus, we can observe positive transfer, as expected, but also hints that expected positive transfer does not occur, which might be linked to idiosyncrasies of the tasks. We dive deeper into this insight in Section V-B.

For the questions where we expected *negative* transfer, we see low (Q1, Q4, Q6) to medium (Q8, Q9) correctness rates in the 2023 instance. This is similar in the 2024 instance, but with slight improvements on the correctness rates. Thus, negative transfer also occurred.

Interestingly, for questions in which we expected *no transfer*, the correctness rate for Q7 is 0 for most students. For Q10, some students could still infer the correct response, even though we did not expect that they can apply their previous knowledge.

2) RQ1.2: Transfer from Other Languages to Java: Now, we describe the observations of transfer from students with a background in Python, in both Python and C, or with no previous programming knowledge.

For students transitioning from Python, we found mixed results. For questions where we expected *positive* transfer, either transfer occurred (Q3, Q5) or it did not (Q6, Q10, in which all Python students in the 2023 instance eventually gave up). In questions with *negative* transfer, the picture is also mixed, with low correctness rates in Q1, Q2, and Q4, but medium to high correctness in Q8. The questions where we expected *no transfer* showed a low (Q7) or medium to low correctness rate (Q9).

For students transitioning from a mixed background of C and Python, we found high correctness rates in Q3 and Q5, where *positive* transfer was expected from both C and Python. For questions in which we expected *negative* transfer, we found low (Q4) and medium (Q8) correctness rates in both instances. For Q1, we observed an increase in correctness in the 2024 instance from low to medium. For the remaining questions, we expected different kinds of transfer for different programming languages, and there is a mixed picture. For some questions, these students outperformed students who are familiar with only one language (e.g., Q6 and Q10), and for others, being familiar with only one language seems to be better for transfer (Q9). One interesting observation is that the performance of these students was better for most tasks compared to students who are familiar with C only. This is especially surprising for the tasks in which we expected a negative transfer from both, Python and C.

For the students who had no programming background, we assume that *no transfer* should happen during their snippet comprehension. This is also reflected in the correctness rates, which are mostly lower than average.

To summarize, we could observe all kinds of transfer when students transition to Java. However, the transfer seems to be less driven by the expected type of transfer, and more by the specifics of the task.

C. RQ2: What is the effect of explicit Java grammar instruction on transfer?

To answer RQ2, we compared data from the 2023 instance with that from the 2024 instance. On average, there was an increase in correctness rate for all tasks in the 2024 instance.

1) RQ2.1: Effect on Transfer from C to Java: For the questions in which we expected *positive* transfer, we observed improvements in the correctness rates of two questions (Q3, Q5), which were already relatively high in the 2023 instance. The correctness rate of Q2 remained low in both instances.

For all questions in which we expected *negative* transfer, we found increases in the correctness rates, with considerable increases in Q6, Q8, and Q9, but only marginal increases for Q1 and Q4.

In the question where we expected *no transfer* (Q10), the correctness rate improved from low to medium.

2) RQ2.2: Effect on Transfer from Other Languages to Java: Unfortunately, the number of students with only Python background is rather low to reason about the effect of a Java tutorial. But one interesting observation is that the difficult questions (i.e., with low correctness rates) Q2, Q4, and Q6 remain difficult in the 2024 instance.

For students with both C and Python backgrounds, we observed improvements on the correctness rates in most questions (Q1, Q2, Q4, Q5, Q6, Q9, Q10) and particularly considerable improvements in Q1 and Q2, in which the correctness rates more than doubled. The correctness rate of Q3, where we expected *positive* transfer from both languages, remained almost the same. In Q8, where we expected *negative* transfer from both languages, the correctness rate decreased slightly.

For students with no programming background, we found improvements on the correctness rates in most questions (Q2, Q3, Q4, Q5, Q8, Q9, Q10) and decreases in two questions (Q1, Q6). Interestingly, the correctness rates increased from 0 to low or medium in three questions (Q2, Q4, Q9), while the correctness rate for Q5 improved from medium to 100 %.

To summarize, not surprisingly, there is a general benefit of introducing basic Java concepts when learning Java. However, there are some concepts that are more susceptible to improve with a tutorial than others. We dive into this into more detail when looking at the errors in more detail in Section V-B.

V. Discussion and Exploration

In general, we could replicate the findings by Tshukudu and Cutts [5] that transfer occurs during the learning of a second programming language, similar to natural language learning. We now discuss these findings in detail, starting with a comparison to the original study, followed by a closer look at how transfer is connected to specific concepts.

A. Comparison with Tshukudu

By analyzing our results, we observed the behavior of students regarding programming language transfer, which led us to findings consistent with Tshukudu and Cutts [5].

For the tasks in which we expected *positive* transfer from both C and Python (Q3, Q5), we found high correctness rates which are similar to the performance of students in the work of Tshukudu [16]. In the 2024 instance, after the Java tutorial, the performance of students improved considerably, indicating that students could benefit from the tutorial when explicitly learning how loops and methods work in Java. However, for the concepts scoping (Q2, C background) and object assignment (Q6, Python background), we found low correctness rates before and after the tutorial, suggesting that these are especially difficult for students to learn and require more than an introduction of these concepts. This is also in line with the results by Tshukudu and Cutts [5].

We observed low correctness rates and no improvement in Q4, in which we expected *negative* transfer. This is, consistent with Tshukudu, who found that 77 % of the students (Python) still struggled with the comparison operator == [16]. There is a similar situation in Q1, which was related to type compatibility. Even though both, Java and C, have a static type system, we observed little improvement after the Java tutorial in the 2024 instance, indicating that students struggled to pick up more nuanced differences between type systems. Thus, it might be helpful for students that we dedicate more time to explicitly explain nuanced differences and similarities between type systems.

By contrast, there was a considerable increase in the correctness rate of Q9 (two-dimensional arrays) after the Java tutorial, as well as in Q6 (attributes of objects) and Q10 (classes) for students with C background, and Q8 (string concatenation), indicating that students could easily improve their understanding in object assignment, string concatenation, and classes. This is in contrast to findings of Tshukudu and Cutts [5], [16]. One explanation could be that these concepts are more similar in C and Java, so C students can more easily grasp them after dedicated explanations.

> Students from C profit differently from dedicated instructions than students from Python. Especially concepts of classes, object assignment, and string concatenation can be more easily grasped after dedicated training. However, the transfer between the type system from C to Java seems to pose troubles for students.

B. Exploring Kinds of Errors per Task

Since the transfer appears to be tightly connected to the specific tasks, we dive deeper into the kind of errors per task. To this end, we conducted an open card sorting and identified categories of errors that students made [40]. Based on these categories, we can get more insights into how transfer took place or failed. We omit Q3 and Q5 because of the high correctness rates. In Table IV and Table V, we included the error categorization for Q1 and Q8 as examples. Due to the page limit, we cannot include the tables for all questions, but they are available on our Web site.

1) Q1: In Q1 (assigning a float number "10.5" to an integer variable whose initial value is 1), we expected *negative* transfer, because C and Python handle type mismatches differently than Java. For students coming from C, we would have ex-

TABLE IV
CATEGORIZATION OF ERRORS IN Q1 IN THE TWO INSTANCES

2023	Type Incompatibility	Type Casting Misconception (1)	Type Casting Misconception (10.5)	Other	Sum	2024	Type Incompatibility	Type Casting Misconception (1)	Type Casting Misconception (10.5)	Other	Sum
C	8	3	25	2	38		3	1	9	0	13
Python	2	0	1	0	3		0	1	1	0	2
C & P	5	1	10	0	16		2	0	6	1	9
None	2	1	3	0	6		3	1	7	1	12
Other	3	0	5	1	9		0	0	1	1	2
Sum	20	5	44	3	72		8	3	24	3	38

pected the responses "10" or "11", because this would be more close to the behavior in C, in which an implicit conversion of the floating point number 10.5 occurs, transforming it to the integer number 10, which is assigned to the variable. For students coming from Python, we would expect the response "10.5", because of the dynamic typing in Python.

However, most students with C background responded with "10.5", so they assumed that it was possible to assign a floating point value to an integer variable without causing an error in Java. Thus, there was *no transfer* of their knowledge of the C type system to Java. Similarly, students who are familiar with both, C and Python, mostly applied their knowledge of Python's type system (10), less so of the C type system (5).

We also observed students responding "10" or "11", indicating that they noticed the type incompatibility and assumed a behavior close to C. Interestingly, these errors persisted even with dedicated Java instructions. Thus, type systems seem to be something that is not straight forward to learn, even when students already are familiar with static typing. A more detailed introduction to the type system of Java and the differences to C, no matter how small they might appear, might help to support transfer, even when students transfer from another statically typed language.

Interestingly, some students seem to have ignored the assignment statement and responded that "1" is the output. They might have noticed the problem with the assignment, ignoring it, and only consider the assignments of which they know the behavior.

2) Q2: We identified cases where the expected transfer did not occur in Q2 (declaring a variable "Hello" inside a for-loop, printing it twice, and accessing it outside the loop). For students coming from C, we expected *positive* transfer, because C and Java treat the scope of variables similarly. However, only few students correctly identified a compiler error, which persisted also after a dedicated introduction to Java. Instead, most students responded three times "Hello" or two times "Hello". This indicates that they have an incorrect assumption about scoping, but the concrete assumptions and their incompatibility with the scoping in Java would require dedicated studies, for example, using a think-aloud protocol.

For students coming from Python, we expected *negative* transfer, so the response three times of "Hello", since variables defined within a loop are accessible outside the loop. This is also what we observed in both instances.

For students with both C and Python backgrounds, a larger number applied their Python knowledge (9) compared to those who applied their C knowledge (3). These students seem to profit from a Java tutorial, as do the students who have no programming experience or experience with other programming languages.

Interestingly, we found a few students with a C background (2) and few students with both, C and Python background (1) in the 2023 instance responded "He" or "hel", which is a part of the string "Hello". They seemed to treat the string character by character, which is how they are usually processed in C. This indicates that these students transferred their C knowledge to Java, but had misconceptions about the string representation, which also seems to be easily fixed with the tutorial.

3) Q4: In Q4 (using the "==" operator to check the reference equality of a and b) on object comparison, we expected *negative* transfer from C and Python, because these languages handle objects differently than Java.

For students coming from C and/or Python, we expected the responses "True"[1], which is what most of the students responded. Thus, all these students assumed that the content of the strings was compared, indicating that they transferred their knowledge.

Interestingly, some other students with a C background did not think the comparison operator could compare strings and answered "No output" or "Error". They seemed to assume that the goal of the Java snippet was to compare the content of both strings. C does not allow one to use the comparison operator "==" to compare two strings, instead `strcmp()` function can be used. Thus, we can assume that these students also transferred their C knowledge to Java.

Some students responded with the string's content as the answer. They seemed to have a misunderstanding of the return value of the comparison operator "==". One possible explanation for their answer is that they compared the content of both strings, determined them to be equal, and output the result of the comparison in the form of the equal content.

4) Q6: In Q6 (one object was assigned to another so they point to the same space in memory, and then the attributes of the objects were accessed), we expected *negative* transfer

[1]In the preceding CS1 course, students used the library `stdbool.h`, so are familiar with `true` and `false` for the type `bool`.

from C, such that students would respond “51 53 51”. Most students coming from C answered that they did not know the answer. Some (8) responded with the expected incorrect answer, demonstrating the negative transfer. This type of question seems to profit from dedicated instructions, as in the 2024 instance, there is a considerable increase in correctness rate for the students coming from C. Thus, demonstrating the similarity of references in C and Java seems to be helpful for students to better transfer their knowledge.

For students coming from Python, we expected the response “51 53 53” since the assignment acts *similarly* in Python compared to that in Java. Surprisingly, most students coming from Python provided the same answer as the C students: “51 53 51”. In this case, they did not apply the knowledge from Python, instead assumed that only the values were copied.

For students with a background in both, C and Python, one third of them correctly answered the question, indicating that they used their Python knowledge. Some (3) also used their C knowledge and responded with “51 53 51”. Interestingly, dedicated instructions do not seem to help with better understanding how variables with complex data types work in Java, but still, students seem to better grasp this concept compared to students with solely knowledge in C.

Interestingly, some students answered “51 51” or “51 51 (I do not know what .agga does)”, indicating that the method call confused them. However, they also could correctly answer a question about method calling (Q5). Thus, the combination of method calling and object use seems to be confusing, so explicitly explaining how both work together might help students to combine their existing knowledge.

5) Q7: Q7 focused on the usage of keyword `new` in Java to allocate new memory space for an array. We expected *no transfer* from C and Python, because no syntax similar to “new” exists to manage the memory space in these languages compared to Java. However, ways of managing the memory exist in both languages. We assumed that students coming from C would point out the problem with `new`, since `malloc` function is often used to manually manage the memory. For students coming from Python, we expected similar cases: although Python has automatic memory management similar to Java, the keyword `new` does not have a match.

From the categorization, we found that most students from all kinds of background answered that they did not know the answer. This indicates that they were not able to apply their previous knowledge to Java, in line with our expectation.

We observed that some students also had difficulties in array indices, array length, for-loops, and increment (`++i`). Thus, with all the different concepts that this question contains, it might be too difficult for students at such an early point in transition, so we removed it in the 2024 instance.

6) Q8: Q8 focused on the string concatenation using “+” sign: two strings were connected (“hello”+“there”), and a string and a number were connected (“exec”+3). We expected *negative* transfer from both, C and Python, because of the differences in usage of “+”.

For students coming from C, we would have expected the response “error” or “compiler error”, because C does not support concatenating two strings with “+” sign, instead, the `strcat()` function is often used. Contrary to this expectation, about a third of the students answered this question correctly, so they assumed that “+” can be used for string concatenation. Only one student responded “Error” as expected. Some (9) students provided “hellothere, error”, which is the expected response for students coming from Python, indicating that they did not transfer their C knowledge. Some students (10) answered “hellothere exe” or “hellothere c”, suggesting that they had a misconception about what “+ 3” does and assumed that it interacted with the amount of characters that would be printed. Similar to our observation in Q2, these students transferred their C knowledge in an interesting way to Java.

For students coming from Python, the expected response is “hellothere, error”, since concatenating two strings in this way is allowed in Python, but not a string and a number. The students either responded as expected, thus transferring their knowledge from Python, or got the answer correct. The same was the case for students with mixed C and Python background.

We also found that some students responded “hellothere exec”, thus ignoring the “+ 3”, and only considering the part they were familiar with. This is similar to the case in discussion of Q1.

Interestingly, even though we expected *negative* transfer, we have a relatively high correctness rates. One possible explanation for this is that, since the snippet was relatively simple and short, students could easily infer the meaning of it and arrived at the correct solution. In this way, the influence of negative transfer was mitigated, leading to a high correctness rate. Regarding explicit instructions, most students seem to profit, especially students who have no experience with previous languages seem to profit in quickly understanding how the “+” operator works. Interestingly, for students who are familiar with C and Python, the dedicated tutorial does not seem to help, indicating that the details of how the “+” operator works requires more attention to ignore what they have previously learned and acquire the new meaning of the “+” operator in Java.

7) Q9: The snippet in Q9 initialized a two-dimensional array and printed its contents using nested for-loops, and the focus was using “.length” to access the length of the array in the conditions of the loops. We expected *negative* transfer from C because of the different use of the dot operator (“.”). We expected *no transfer* from Python, because Python has a distinct way of accessing the length of an array with the `len()` function.

Independent of their background, most students responded “I do not know”, indicating that they cannot find any similarity between their familiar syntax and Java syntax. Some students with both C and Python backgrounds stated more specifically in their responses that they did not know whether “.length” was available in Java, which is coherent with our expectation.

TABLE V
CATEGORIZATION OF ERRORS IN Q8 IN THE TWO INSTANCES

2023	+ Concatenation Knowledge Missing	Strings and Numbers Connection Misconception	"+3" misunderstanding	Other	Sum	2024	+ Concatenation Knowledge Missing	Strings and Numbers Connection Misconception	"+3" misunderstanding	Other	Sum
C	1	9	10	3	23		0	3	5	0	8
Python	0	2	0	0	2		0	0	0	0	0
C & P	0	6	0	1	7		0	3	2	3	8
None	0	0	2	1	3		1	0	2	1	4
Other	0	2	0	0	2		0	1	0	1	2
Sum	1	19	12	5	37		1	7	9	5	22

Also, some students from different backgrounds answered Q9 correctly. They seemed to have been able to infer that accessing the length array could be achieved using ".length" in Java. The correctness rates also drastically improved in the 2024 instance, especially for students with a background in C. Dedicated Java instructions seemed to help them in understanding the usage of ".length" and applying their knowledge to a new programming language, which counts for students independent of their background (except Python).

Moreover, we observed that some students understood that the goal of the snippet was to print each element of the array. However, they struggled with the nested for-loops and arrived at incorrect answers, suggesting that nested loops in combination with array access poses difficulties when transferring to Java.

8) Q10: Q10 focused on the basic concepts of classes, and information was printed using getter methods. We expected *no transfer* from C, because C does not have classes, so students would express confusion. We expected *positive* transfer from Python, since Python has classes and similar keywords (class, get, set) are also used.

Most students with a C background said they did not know the answer, indicating that they did not find related knowledge in C, as we expected. Surprisingly, none of the Python students provided a correct solution in neither of the instances, so somehow, classes pose an obstacle for students coming from Python. Interestingly, some students with both C and Python backgrounds also did not know the answer, indicating that they failed to transfer their Python knowledge in this case.

We observed that some students with a C background arrived at the correct answer, especially when they received the dedicated Java instructions. They seemed to have transferred their knowledge of `structs` in C to the unfamiliar but equivalent representation in Java. All students (except the ones coming from Python) profit considerably from explicitly introducing classes, helping them better understand how classes work in Java.

There were a few instances where students with a background in C or mixed backgrounds responded with a part of the correct output, for example "Max25" or "001 Max 25 001 Max 23". These students might have had trouble with method calls, similarly to question Q6, or they might have ignored parts of the code they were not familiar with, similarly to what happened in Q1 and Q8.

Task-specific effects affect transfer differently than expected and cannot be mitigated by dedicated instructions. Especially type incompatibility, variable scoping, and object comparison seem to pose difficulty to students, while string concatenation and classes seem to be easier to grasp once we explicitly demonstrated similarities. This counts also for students who have no programming experience, which might also indicate some kind of transfer, for example, based on previous language learning or real-world objects.

C. Interpreting Unexpected Errors

Our deep-dive into the specific answers shows that transfer does not always occur in the way we expected. In Q1, we expected negative transfer from C and the correctness rates were low, as expected, but the specific answers reveal that students often answered wrong in a way that is not explainable with the background in C. This implies that there was, in fact, not *negative* transfer from C, but *no* transfer.

In other questions, the influence from C is evident, illustrated well by Q2 and Q8, where strings were involved. Some answers indicate that the students tried to handle the strings character by character, demonstrating negative transfer from C. Thus, transfer from previously known languages does occur, but depends on further factors.

One possible factor affecting transfer is a mix of several concepts within the same code snippet. Even the seemingly simple snippets sometimes still contain more than one concept. Q2 is a good example again: The question focuses on variable scope, and we expected positive transfer. However, the students who thought that the code would print characters of the string clearly did not transfer their knowledge on scoping from C, but they did transfer their knowledge on strings. In other words, they did not transfer their knowledge on the expected concept, but they did on another concept that was also contained within the snippet. Another example is Q6. We expected to observe negative transfer on the concept of object assignment, and the correctness rate was appropriately low. However, many students stated that they did not know the

answer, as opposed to the answer we expected due to negative transfer from C. This also looks like there was no transfer, and some answers indicate that the concept of methods could have been an issue. This is especially interesting, because in Q5, most students demonstrated that they can correctly identify a method call in Java. However, they could not repeat this in Q6, when it was combined with other concepts.

Another phenomenon that we observed is students apparently skipping 'problematic' statements (Q1, Q2, Q6, Q8, Q10). These examples could indicate a tendency in the behavior of the students to strive for the most meaningful or most complete answer, even if they do not understand every part of the snippet. Furthermore, students could avoid providing "error" as an answer, which might be one reason for some concepts not being transferred as expected, such that a concept could be ignored when it leads to "error" as the answer. Future studies could investigate whether erroneous code snippets actually lead to different behavior when answering comprehension questions, and whether the phenomenon is tied to transfer or more ubiquitous.

To summarize, the analysis of the errors revealed that students often did not answer the way we predicted based on the Model of Programming Language Transfer [5]. The concrete answers hint at further possible factors that are at play here, such as several different concepts being contained in one code snippet. Thus, transfer does take place, but it is unclear why some concepts transfer while others do not, or which concept or knowledge source takes precedence when several are at play.

Tshukudu also suspected that the "distance" between the concepts in different languages could be an influence, that is, some concepts are more different between the compared languages than others. Thus, explicitly bridging this distance, for example, by explaining how structures map to classes, might reduce this distance, while for others, the distance might simply be too big to close in one tutorial. This distance could also be perceived differently by different persons: While instructors might feel that type handling is very similar, students might feel that the difference is larger, so they fail to see the similarity. [16]

VI. Threats to Validity

Like every empirical study, there were some threats to validity that we could not entirely exclude. A threat to *construct validity* is whether we truly observed transfer or other effects. To mitigate this threat, we stuck as closely as possible to the established snippets and carefully examined each snippet for the expected kind of transfer. The existence of the expected incorrect responses shows that the snippets indeed let students transfer their knowledge, but the unexpected incorrect responses demonstrate that there might be something more. Still, the results provide interesting insights into how students learn a new programming language. Furthermore, the dedicated Java tutorial might have omitted concepts of Java or not went into too much depth. To mitigate this threat, we focused on concepts of the snippets, and we discussed the results per task, identifying specific problems. Thus, our results provide relevant avenues on how future work could address especially problematic concepts.

Regarding *internal validity*, the snippets could lead to either an error or a certain output. To ensure that participants would feel confident to give these two kinds of responses, we explicitly told participants that an error is also a possible response. Furthermore, we anonymized the responses of participants, so that they would not feel evaluation apprehension. Additionally, we could not determine the previous experience of participants. To take this into account, we assessed it with an established questionnaire and discussed the results separately by different programming languages.

For *external validity*, we focused on students of a Java CS2 course. Thus, the results do not appear very general, but they are a piece in the puzzle on understanding learning transfer.

VII. Conclusion

Being able to quickly learn new programming languages is an essential skill for software developers. In our study, we have evaluated the difficulties students face when transitioning from the C programming language to Java, thereby providing further empirical evidence that transfer in programming learning is similar to that in learning a second natural language. Explicit instructions about differences and similarities between programming languages can foster transfer, but some concepts, even though they appear similar, pose considerable obstacles. These insights help us to tailor programming teaching in more advanced programming courses, as we can better understand the obstacles students face, on which we can put special focus.

Data Availability

All data and supplementary material is available at: https://github.com/CodePenguinny/Knowledge-Transfer-and-False-Friends.git.

Acknowledgment

We like to thank our participants. This work was inspired by Dagstuhl Seminar XXXXX.

References

[1] R. S. N. Lindberg, T. H. Laine, and L. Haaranen, "Gamifying programming education in k-12: A review of programming curricula in seven countries and programming games," *British Journal of Educational Technology*, vol. 50, no. 4, pp. 1979–1995, 2019. [Online]. Available: https://bera-journals.onlinelibrary.wiley.com/doi/abs/10.1111/bjet.12685

[2] E. Commission, E. Education, and C. E. Agency, *Informatics education at school in Europe*. Publications Office of the European Union, 2022. [Online]. Available: https://data.europa.eu/doi/10.2797/268406

[3] H. Belmar, "Review on the teaching of programming and computational thinking in the world," *Frontiers in Computer Science*, vol. 4, 2022. [Online]. Available: https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2022.997222

[4] D. N. Perkins, G. Salomon *et al.*, "Transfer of learning," *International encyclopedia of education*, vol. 2, pp. 6452–6457, 1992.

[5] E. Tshukudu and Q. Cutts, "Understanding conceptual transfer for students learning new programming languages," in *Proceedings of the 2020 ACM conference on international computing education research*, 2020, pp. 227–237.

[6] K.-C. Lu, S. Krishnamurthi, K. Fisler, and E. Tshukudu, "What happens when students switch (functional) languages (experience report)," *Proc. ACM Program. Lang.*, vol. 7, no. ICFP, aug 2023. [Online]. Available: https://doi.org/10.1145/3607857

[7] R. K. Runde, Q. Cutts, and L. K. Skaarseth, "Exploring scratch to python transfer in norwegian lower secondary schools," in *Norsk IKT-konferanse for forskning og utdanning*, no. 4, 2023.

[8] O. Karnalim, M. Ayub, M. C. Wijanto, and F. Hermans, "Does hedy, the gradual programming language help computing undergraduates to learn programming?" in *Towards a Hybrid, Flexible and Socially Engaged Higher Education*, M. E. Auer, U. R. Cukierman, E. Vendrell Vidal, and E. Tovar Caro, Eds. Cham: Springer Nature Switzerland, 2024, pp. 187–198.

[9] E. Tshukudu and Q. Cutts, "Semantic transfer in programming languages: Exploratory study of relative novices," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 307–313. [Online]. Available: https://doi.org/10.1145/3341525.3387406

[10] E. Tshukudu, Q. Cutts, and M. E. Foster, "Evaluating a Pedagogy for Improving Conceptual Transfer and Understanding in a Second Programming Language Learning Context," in *Proceedings of the 21st Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3488042.3488050

[11] J. Holvitie, T. Rajala, R. Haavisto, E. Kaila, M.-J. Laakso, and T. Salakoski, "Breaking the programming language barrier: Using program visualizations to transfer programming knowledge in one programming language to another," in *2012 IEEE 12th International Conference on Advanced Learning Technologies*, 2012, pp. 116–120.

[12] J. Strømmen, "How to provide automated feedback helping students with negative semantic transfer when learning a second programming language," Master's thesis, The University of Bergen, Jun. 2022, accepted: 2022-07-07T23:41:10Z. [Online]. Available: https://bora.uib.no/bora-xmlui/handle/11250/3003710

[13] P. Denny, B. A. Becker, N. Bosch, J. Prather, B. Reeves, and J. Whalley, "Novice reflections during the transition to a new programming language," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, ser. SIGCSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 948–954. [Online]. Available: https://doi.org/10.1145/3478431.3499314

[14] J. Spilling, "Investigating intermediate student transfer to functional programming," Master's thesis, University of Oslo, 2023.

[15] Z. Li, S. Yang, K. Cunningham, and A. Alawini, "Assessing student learning across various database query languages," in *2023 IEEE Frontiers in Education Conference (FIE)*, 2023, pp. 1–9.

[16] E. Tshukudu, "Understanding Conceptual Transfer in Students Learning a New Programming Language," Ph.D. dissertation, University of Glasgow, 2022.

[17] D. Parsons and P. Haden, "Programming osmosis: Knowledge transfer from imperative to visual programming environments," in *Procedings of The Twentieth Annual NACCQ Conference*. Citeseer, 2007, pp. 209–215.

[18] K. Powers, S. Ecott, and L. M. Hirshfield, "Through the looking glass: teaching cs0 with alice," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 213–217. [Online]. Available: https://doi.org/10.1145/1227310.1227386

[19] D. Weintrop and U. Wilensky, "Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms," *Computers & Education*, vol. 142, p. 103646, Dec. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S036013151930199X

[20] C. V. Alejandro Espinal and V. Guerrero-Bequis, "Student ability and difficulties with transfer from a block-based programming language into other programming languages: a case study in colombia," *Computer Science Education*, vol. 33, no. 4, pp. 567–599, 2023. [Online]. Available: https://doi.org/10.1080/08993408.2022.2079867

[21] D. Sun, C. Zhu, F. Xu, Y. Li, F. Ouyang, and M. Cheng, "Transitioning from introductory to professional programming in secondary education: Comparing learners' computational thinking skills, behaviors, and attitudes," *Journal of Educational Computing Research*, vol. 62, no. 3, pp. 647–674, 2024. [Online]. Available: https://doi.org/10.1177/07356331231204653

[22] M. Mladenović, Žana Žanko, and G. Zaharija, "From blocks to text: Bridging programming misconceptions," *Journal of Educational Computing Research*, vol. 0, no. 0, p. 07356331241240047, 0. [Online]. Available: https://doi.org/10.1177/07356331241240047

[23] M. Kölling, N. C. C. Brown, and A. Altadmri, "Frame-based editing: Easing the transition from blocks to text-based programming," in *Proceedings of the Workshop in Primary and Secondary Computing Education*, ser. WiPSCE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 29–38. [Online]. Available: https://doi.org/10.1145/2818314.2818331

[24] P. E. C. Barata, J. a. V. P. Corrêa, and M. P. Mota, "A study on knowledge transfer between programming languages by programs meanings facets," in *Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems*, ser. IHC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3160504.3160530

[25] S. Grover, "Teaching and Assessing for Transfer from Block-to-Text Programming in Middle School Computer Science," in *Transfer of Learning: Progressive Perspectives for Mathematics Education and Related Fields*, C. Hohensee and J. Lobato, Eds. Springer International Publishing, 2021, pp. 251–276. [Online]. Available: https://doi.org/10.1007/978-3-030-65632-4_11

[26] D. Krpan, S. Mladenović, and G. Zaharija, "Mediated transfer from visual to high-level programming language," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2017, pp. 800–805.

[27] J. McKenna and Y. Lin, "Switch mode: A tool for transitioning students from block-based to text-based programming," in *Proceedings of Society for Information Technology & Teacher Education International Conference 2024*, J. Cohen and G. Solano, Eds. Las Vegas, Nevada, United States: Association for the Advancement of Computing in Education (AACE), March 2024, pp. 1181–1184. [Online]. Available: https://www.learntechlib.org/p/224111

[28] M. Branthôme, "Pyrates: A serious game designed to support the transition from block-based to text-based programming," in *Educating for a New Future: Making Sense of Technology-Enhanced Learning Adoption*, I. Hilliger, P. J. Muñoz-Merino, T. De Laet, A. Ortega-Arranz, and T. Farrell, Eds. Cham: Springer International Publishing, 2022, pp. 31–44.

[29] M. Kazemitabaar, V. Chyhir, D. Weintrop, and T. Grossman, "Codestruct: Design and evaluation of an intermediary programming environment for novices to transition from scratch to python," in *Proceedings of the 21st Annual ACM Interaction Design and Children Conference*, ser. IDC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 261–273. [Online]. Available: https://doi.org/10.1145/3501712.3529733

[30] C. Izu and C. Mirolo, "Learning transfer in novice programmers: A preliminary study," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, 2021, pp. 178–184.

[31] D. Teague and R. Lister, "Manifestations of preoperational reasoning on similar programming tasks," in *Proceedings of the Sixteenth Australasian Computing Education Conference [Conferences in Research and Practice in Information Technology, Volume 148]*. Australian Computer Society, 2014, pp. 65–74.

[32] D. Ginat, E. Shifroni, and E. Menashe, "Transfer, cognitive load, and program design difficulties," in *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 2011, pp. 165–176.

[33] S. Garcia-Martinez and D. Zingaro, "Teaching for transfer of learning in computer science education," *Journal for Computing Teachers*, pp. 1–6, 2011.

[34] F. Hermans, "Hedy: A gradual language for programming education," in *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ser. ICER '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 259–270. [Online]. Available: https://doi.org/10.1145/3372782.3406262

[35] J. Scholtz and S. Wiedenbeck, "Learning second and subsequent programming languages: A problem of transfer," *International Journal of Human–Computer Interaction*, vol. 2, no. 1, pp. 51–72, 1990. [Online]. Available: https://doi.org/10.1080/10447319009525970

[36] N. Shrestha, T. Barik, and C. Parnin, "It's like python but: Towards supporting transfer of programming language knowledge," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018, pp. 177–185.

[37] N. Shrestha, C. Botta, T. Barik, and C. Parnin, "Here we go again: why is it difficult for developers to learn another programming language?" in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 691–701. [Online]. Available: https://doi.org/10.1145/3377811.3380352

[38] Y. Ma and E. Tilevich, ""you have said too much": Java-like verbosity anti-patterns in python codebases," in *Proceedings of the 2021 ACM SIGPLAN International Symposium on SPLASH-E*, ser. SPLASH-E 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 13–18. [Online]. Available: https://doi.org/10.1145/3484272.3484960

[39] J. Siegmund, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and modeling programming experience," *Empirical Software Engineering*, vol. 19, pp. 1299–1334, 2014.

[40] W. Hudson, "Card Sorting," in *The Encyclopedia of Human-Computer Interaction*, I. D. Foundation, Ed. Interaction Design Foundation, 2013.