Sagol School of Neuroscience

Department of cognitive neuroscience

How working memory supports programming and does programming support it back?

By

Lihi Catz

208481119

The thesis was carried out under the supervision of Dr. Dror Dotan

September 2023

Table of contents

Abstract

The importance of working memory (WM) in executing mental algorithms is well established, but we still do not understand the precise bi-directional relations between specific WM processes and specific aspects of mental algorithms, i.e., how precisely WM impacts the execution of mental algorithms, and whether and how 'mental algorithm training' improves WM. We examined these issues in two experiments that used a novel task focusing on simple computer programming skills.

We centered on a specific WM mechanism – the "Focus of Attention (FOA)", the WM region that contains one information item selected for the current cognitive operation. Previous studies showed that shifting the FOA from one item to another entails a cognitive cost; here, we examined how this applies to programming. Participants followed, in their heads, simple code snippets that set the value of 2-3 variables and then updated them: a series of commands was presented one at a time, and each command updated the values of a single variable. Similar to findings from other domains, we found a switch cost: when a command updated a variable different from the previous command, it took longer, and caused more errors, than commands without a variable-switch – presumably because of the cost involved in shifting the focus of attention from one variable to another.

To see the reverse causality, namely how training in the programming task affects WM, we examined whether and how the participant's performance improved during the experiment session. In Experiment 1, the performance (errors, reaction times) improved throughout the session, and importantly, the improvement was larger in the variable-switch trials than in the no-switch trials. We conclude that the improvement was not a general improvement in the task, and not even a general improvement in vaguely-defined WM skills, but it was specifically in the FOA-shifting mechanism. Experiment 2, which used a similar method in a slightly different configuration, failed to replicate this finding. We propose that the reason for this discrepancy may be that specific aspects of the training, which were present in Experiment 1 but not in Experiment 2, may have been critical to the effectiveness of the training; and we suggest how future studies may use these discrepancies to identify the factors that are critical for effective WM training.

We also found that the variables that were **not** used in the current trial ("inactive variables"), and were stored in WM region/s with lower accessibility level than the FOA, were not all stored with the same accessibility levels. To assess this, we examined how the performance in a particular variable is affected by its "inactivity duration" – the number of trials elapsed since it was last used. The performance was poorer in trials with longer inactivity duration, indicating that longer inactivity reduced the variable's accessibility level. We discuss the implications of this finding to our understanding of how the inactive variables are represented in WM.

Overall, the study shows that our novel paradigm can capture the bi-directional causal relations between specific WM processes and specific aspects of executing programming algorithms mentally.

תקציר

החשיבות של זיכרון פעיל בביצוע אלגוריתמים מנטליים ידועה והודגמה במחקרים רבים, אבל אנו עדיין לא מבינים באופן מדויק את היחסים הדו-כיוונים בין תהליכי זיכרון פעיל ספציפיים והיבטים ספציפיים של אלגוריתמים מנטליים. כלומר, כיצד בדיוק זיכרון פעיל משפיע על ביצוע אלגוריתמים מנטליים, והאם ואיך בדיוק אימון בביצוע אלגוריתם מנטלי משפר את הזיכרון הפעיל. המחקר הנוכחי בדק את הנושאים הללו בשני ניסויים בעזרת מטלה חדשה, שפותחה כאן, שבחנה מיומנויות פשוטות של תכנות.

התרכזנו במנגנון ספציפי של זיכרון פעיל – "מוקד הקשב", האזור בזיכרון הפעיל שמכיל פריט מידע אחד שנבחר עבור הפעולה הקוגניטיבית הנוכחית. מחקרים קודמים הראו שהעברת מוקד הקשב מפריט אחד לאחר כרוכה ב״מחיר״ קוגניטיבי; כאן, בדקנו איך זה משפיע על תכנות. המשתתפים עקבו בראשם אחר קטעי קוד פשוטים, שתחילה קבעו את הערך של 2-3 משתנים ואז עדכנו אותם: סדרה של פקודות הוצגו למשתתפים זו אחר זו, וכל פקודה עדכנה את הערך של משתנה אחד. בדומה לממצאים מתחומים אחרים, מצאנו שהיה ״מחיר״ להחלפת משתנה (switch cost): כאשר המשתנה בפקודה הנוכחית היה שונה מהמשתנה בפקודה הקודמת, הפעולה לקחה יותר זמן, והיו בה יותר שגיאות, מאשר פקודות ללא החלפת משתנה – ככל הנראה בגלל העלות הכרוכה בהסטת מוקד הקשב ממשתנה אחד לאחר.

כדי לראות את הכיוון ההפוך של הסיבתיות, כלומר כיצד אימון במטלת התכנות משפיע על הזיכרון הפעיל, בדקנו האם וכיצד השתפרו הביצועים של המשתתפים במהלך מפגש הניסוי. בניסוי 1, הביצועים (שגיאות, זמני תגובה) השתפרו לאורך המפגש, וחשוב מכך, השיפור היה גדול יותר בצעדים עם החלפת משתנה, לעומת צעדים ללא החלפה. הסקנו שהשיפור לא היה שיפור כללי במטלה ואפילו לא בכל יכולות הזיכרון, אלא שיפור ספציפי באותו מנגנון זיכרון פעיל שמסיט את מוקד הקשב. ניסוי 2, שהשתמש בשיטה דומה אך בתצורה מעט שונה, לא הצליח לשחזר את הממצא הזה. אנו מציעים שהסיבה לאי ההתאמה עשויה להיות שהיבטים ספציפיים של האימון, שהיו קיימים בניסוי 1 אך לא בניסוי 2, עשויים להיות קריטיים ליעילות האימון; ואנו מציעים כיצד מחקרים עתידיים יכולים להשתמש באותם פערים שמצאנו בין שני הניסויים כדי לזהות את הגורמים הקריטיים לאימון זיכרון יעיל של זיכרון פעיל.

בנוסף, מצאנו שהמשתנים בהם לא נעשה שימוש בצעד הנוכחי ("משתנים לא פעילים"), שנשמרים באזורי זיכרון פעיל עם רמת נגישות נמוכה יותר מאזור מוקד הקשב, לא אוחסנו כולם באותה רמת נגישות. כדי לבחון זאת, בדקנו כיצד רמת הביצוע במשתנה מסוים מושפעת מ"משך חוסר הפעילות" שלו – מספר הצעדים שחלפו מאז השימוש האחרון באותו משתנה. הביצועים היו גרועים יותר בצעדים עם משך חוסר פעילות ארוך יותר, מכאן שחוסר-פעילות ארוך יותר הפחית את רמת הנגישות של המשתנה. בדיון, אנו מפרטים את ההשלכות של ממצא זה על ההבנה שלנו כיצד מיוצגים המשתנים הלא פעילים בזיכרון הפעיל.

לסיכום, המחקר מראה שהפרדיגמה החדשה שהצגנו כאן יכולה להראות את הקשרים הסיבתיים הדו-כיוונים בין תהליכי זיכרון פעיל ספציפיים לבין היבטים ספציפיים של ביצוע אלגוריתם מנטלי של תכנות.

1. Introduction

1.1. What is working memory?

1.1.1. Which information is stored in WM?

Working memory (WM) is generally described as a cognitive system that stores information temporarily and allows to manipulate the information for complex cognitive operations such as problem solving, learning, executing mental algorithms (as when calculating), reading, etc. (Baddeley, 1992).

Different researchers proposed different definitions to the concept of WM, reflecting different theoretical framing of WM (Cowan, 2017). The present investigation relies primarily on Oberauer's (2002, 2009) definition of WM as a mechanism that provides access to representations of information for goal-directed processing. WM holds the information most relevant to the current cognitive task at any moment, and its content changes rapidly: new, relevant information enters WM, and already-used information, which has become irrelevant, exits WM. To explain the mechanisms underlying these changes, several researchers (see review in Lewis-Peacock et al., 2018) argue that once information was coded in the WM, it is retained by default, without the need for an active processing to maintain it. There is some degree of spontaneous decay, but it is relatively slow. Thus, no-longer-needed information must be removed actively from WM, so it can be cleared out in the appropriate pace and "make room" for new, relevant information (Oberauer, 2020). Formally, removal is defined as the exclusion of information from the working memory, to accomplish the current goal.

1.1.2. Levels of representation in WM

Oberauer's (2002, 2009) model of WM describes 3 functionally distinct regions, each characterized by a different level of accessibility to the information retained there. In increasing order of accessibility, they are the Activated part of the Long-Term Memory (ALTM), the Region of Direct Access (RDA), and the Focus of Attention (FOA).

Activated Long-Term Memory (ALTM). The long-term memory (LTM) contains representations of numerous information items connected via a network of associated representations. This network allows the information items to activate each other and may also be triggered by external perceptual input. The ALTM is the subset of LTM representations currently activated, which are

relevant to the current situation and goal. Of the 3 WM regions, The ALTM has the lowest degree of accessibility and it can store the largest number of information items.

Region of direct access (RDA). The RDA holds a small number of information items (up to 3-4; Cowan, 2001) that are accessible for cognitive manipulations by various cognitive processes. Because of the RDA's limited capacity, the removal-from-WM process (Lewis-Peacock et al., 2018) is highly relevant here. Information can be (and typically is) removed from the RDA when it becomes no longer relevant to the current cognitive task. The removed information still remains in the ALTM; if it becomes relevant again, it will be re-retrieved (Oberauer, 2020).

Each information item in the RDA (e.g., word, object) can be bound to a specific context, or "role". For example, when calculating 7 - 3, the 7 and the 3 would be bound to the minuend and subtrahend roles, respectively. Thus, the RDA information items are not an arbitrary set, they can be integrated into a meaningful structure. The RDA's limited capacity limits the number of roles to which information items can be bound, and consequently the complexity of the structural representations that can be created in WM. Moreover, once information items were bound to a structure, they can remain in this structure even if they are removed to the ALTM; i.e., a structured group of information items can be retrieved from the ALTM to the RDA as a whole, not item by item (Oberauer, 2005).

Focus of Attention (FOA). The FOA holds the information item that is the object of the current cognitive task. This information item has a special status: it can be accessed faster, and often with higher accuracy, than the information items in the RDA. The FOA typically holds one information item, but it can also hold more information items simultaneously if they are bound to the same context/role – i.e., if the task requires simultaneous access to several information items and does not require binding each of them to a different context. The FOA cannot hold more than one context-binding simultaneously (Oberauer, 2020).

Changing the contents of the FOA, i.e., shifting the focus of attention from one information item to another, entail an *object-switch cost* (Garavan, 1998; Oberauer, 2002) in terms of reaction time and accuracy. The switch cost is a function of the difficulty of the selection-from-RDA process, e.g., it increases when there are more information items in the RDA (the *selection interpretation*; Oberauer, 2002).

When selecting an information item to be brought to the FOA, all RDA information items are candidates – a competition that will end with one item being selected. This is called *crosstalk* (Oberauer, 2002). The FOA can retrieve information items only from the RDA, i.e., the ALTM is

not a part of the crosstalk. To recall an information item from the ALTM, it must first be brought to the RDA and bound to the structure via which it can be selected to the FOA.

Previous research used various methods to show the existence of FOA. One is the aforementioned switch cost (Oberauer, 2002). Using a different approach, Garavan (1998) asked participants to count two types of shapes that appeared on the screen, i.e., the participants had to maintain two running 'mental counters' simultaneously. He found that the participants could only attend to one counter at the time: they could not access the two counters in the WM with equal speed. A different method aimed to show the FOA's limited capacity (McElree, 2001): participants had to respond to one item in each trial; when, prior to the response, their attention was focused on one item, performance was better than when their attention was focused on 3 items simultaneously. McElree concluded that in the former case, the single item was already in the FOA, whereas in the latter case the FOA could not hold all 3 attended items.

1.2. The present study

The present study aimed to examine the process of shifting the focus of attention. Specifically, we had two goals. First, we examined how the FOA-shifting would be exhibited in a simple computer programming task – a kind of task in which the effect of WM mechanisms was not tested previously. Specifically, we aimed to capture programming scenarios that entail an FOA switch cost. Second, we examined the possibility of training the FOA-shifting mechanism. This could be interesting, because better WM functions may lead to better performance in many tasks, including perhaps calculating, reading, problem solving, learning etc. Indeed, WM is highly correlated to measures of fluid intelligence (Yuan et al., 2006).

2. Experiment 1

2.1. Method

2.1.1. Participants

There were 32 adults (aged 18-55) with normal or corrected-to-normal vision, Hebrew as native tongue, and no reported cognitive disorders. All participants reported having a minimal background in programming – 2-3 university courses. Both experiments 1 and 2 were approved by the Tel Aviv University Institutional Review Board, and in both the participants were compensated for participation.

2.1.2. Stimuli

The experiment included 40 Python code snippets, each with 12-17 lines of code. The first lines initialized the values of a variable (e.g., "b = 5"), subsequent lines updated it (e.g., "b += 3"). The code snippets appeared on screen one line (command) at a time. We hereby consider each command as one trial. An exception was the first 2-3 lines in each code snippet, which set the initial variable value; these lines appeared on screen simultaneously and were not analyzed.

Each code snippet included 2-3 variables, whose value was always numerical and remained between 1-20 throughout the code snippet. The variable names were consecutive letters (a-b-c, x-y-z, p-q-r, or g-h-k in different code snippets, counterbalanced).

Code snippet structure. The first trial in each code snippet included 2-3 *variable initialization* commands (e.g., "a=3"). The variable values were different from each other, all in the range 4-16. The 2-3 commands appeared one above the other, in alphabetical order of the variable names. Next, there were 8-12 *update* trials, each with a single line of code that updated one of one variable by adding (e.g., "a += 10") or subtracting a fixed value ("a -= 5"). The added/subtracted value was in the range 2-9, and the new value was always in range 1-20. Finally, there were two *print* trials, each with one line of code that printed an expression in the form of an inline-if command that included some of the code snippet's variables. These two print commands were not analyzed.

Experimental conditions. We manipulated two factors. First, the number of variables in each code snippet (2 or 3). Second, the congruency between each update command and the previous update command: in no-switch trials, the trial updated the same variable as the previous one. In switch trials, it updated a different variable. The congruency of the first update trial in each code snippet

was undefined, so these were excluded from congruency analyses. Below, we use the term "switch" to indicate the change in the visual stimulus relative to the previous stimulus, and the term "shift" to indicate the cognitive operation, namely the participant focusing on the mental representation of a variable different from the previous one. We predicted higher error rates and longer reaction times in the switch trials compared to the no-switch trials.

Creating the code snippets. To create the 40 code snippets, we created 10 "core code snippets" and derived 4 variants from each. Each core code snippet had 3 variables. In 4 of the core code snippets, there were 4 update trials for each variable. We derived 4 variants from each of these code snippets: two variants in which the update lines were shuffled in random order, resulting in many switch trials; and two variants in which the trials were grouped by variable (e.g., all updates of b, then all updates of a, then all updates of c), resulting in many non-switch trials and only a few switch trials. In 6 additional core code snippets, there were 4 update trials for two of the variables and 2 update trials for the third variable. From each of these core code snippets, we created two variants as described above (one variant with many switch trials, one with few switch trials); and two more variants, again a many-switches one and a few-switches one, in which we deleted all lines referring to one of the variables (the one with only two update commands), resulting in a code snippet with only 2 variables.

For each participant, we randomized the order of code snippets and the variable-set assigned to each specific code snippet (a-b-c, x-y-z, p-q-r, or g-h-k). The order of code snippets was random, with the limitation that two variants of the same core code snippet were separated by at least 4 other code snippets.

Table 1 shows an example for a code snippet having this structure (for simplicity, this example has only 5 update commands).

Table 1. Sample code snippet consisting of 6 trials. Like all code snippets, it begins with a trial that initializes the variables. Next, there is a series of update trials, each of which updates one of the variables by adding or subtracting a fixed value. The "pre-trial values" column shows the variables before executing the current trial, assuming that all previous trials were answered correctly.

Trial #	Stimulus	Correct response	Pre-trial values	Trial type
0	a = 6 b = 12 c = 3			Initialization
1	a += 8	14	a = 6, b = 12, c =3	Update
2	b -= 4	8	a = 14, b = 12, c = 3	Update (switch)
3	c += 9	12	a = 14, b = 8, c = 3	Update (switch)
4	a +=4	18	a = 14, b = 8, c = 12	Update (switch)
5	a -= 3	15	a = 18, b = 8, c = 12	Update (no switch)

2.1.3. Procedure

The participants were tested individually. The code snippets appeared on screen, one trial at a time, with no time limit. In the first trial in each code snippet, the variable-initialization trial, the participant was asked to memorize the variables and their values and hit the spacebar to continue. In each of the next trials, the participant was to compute the relevant value (the new variable value in case of an update command, or the printed value in case of a print command) and press the corresponding key 1-20 on the keyboard (the digits 1-9, the "0" key for 10, and the keys below them, qwertyuiop, for 11-20). The participant could also respond that the result was smaller than 1 ('tab' key) or larger than 20 ('[' key) – this was never the case in any code snippet, but it could happen if the participant made an error. The participants were instructed that if they made an incorrect response and figured it out only after hitting the response key, they should continue the mental simulation according to the response that they already made for that variable.

In some cases, two subsequent update trials were identical. In retrospect we noticed that these trials caused a confusion, because sometimes the participant thought that the experiment software did not get their response, so they hit the same response key again. Thus, if the trial was identical to the previous one, it was excluded from the analyses.

After each code snippet, a separator screen appeared, instructing the participant to hit the spacebar to start the next code snippet. The participants were allowed to take breaks at this time, and were instructed by the experimenter to take a forced ~1 min break every few minutes.

2.1.4. Error coding

2.1.4.1. *Accuracy*

The coding of accuracy was in accord with our instruction to the participants that they should not try to correct errors that they already made. To accommodate this instruction, we considered the participants' responses in the previous trials. For example, if the participant gave the incorrect answer 6 to a trial that updated variable x, we assumed that the value of x was now 6 (until it was updated again).

If the expected response was within the 1-20 range and the participant responded '< 1' or '> 20', this was coded as an error. If the expected response was outside the 1-20 range (due to errors made in previous trials), the trial was excluded from all analyses, because we could not always tell whether the participant's response was correct or not. We also excluded trials in which the

variable's pre-trial value was ambiguous (this was the case if the participant responded '< 1' or '> 20' to the previous trial that updated this variable).

2.1.4.2. *Error classification*

To classify errors, we defined 14 specific types of errors, such that each error type predicts a specific response in each specific situation. For example, one error type was "+1", i.e., adding an excessive 1 to the calculation. For the trial "x += 8", if the preceding value of x was 2, the correct response is 10. If a "+1" error had been made, the response would be 11. Note that a trial could be classified into more than one error type. For example, if the same code snippet included, on top of variable x, variable y whose current value was 3, the participant's response also matches the possibility of a variable-confusion error, namely adding 8 to y instead of to x.

For each error type, we computed the rate of errors that matched this error type out of the trials in which this error could potentially occur and be detected. For example, the "+1" error cannot be detected if the expected response is 20, because the response predicted by this error (21) entails that the participant would respond by pressing the "> 20" key, and the accuracy of that trial would be ambiguous. Thus, trials whose correct response was 20 were excluded when computing the percentage of the "+1" error. However, these trials could be included when computing the percentage of other error types, e.g., the "-1" error, for which we excluded – for a similar reason – the trials whose correct response was 1.

Below we describe each error type, with an example referring to the code snippet in Table 1. We also define the criteria for trials in which the error type can potentially occur; the number of these trials is the denominator in the calculation of the percent of errors of this type. An additional criterion was that, as explained above, the specific response predicted by the error type must be within the range 1-20. This criterion applies to all error types, so we do not reiterate it for each error type.

The 14 error types were grouped to memory errors, calculation errors, and other errors.

2.1.4.2.1 Memory errors

Memory errors cannot occur on the first trial of a code snippet.

Failed to update: the calculation was done on the previous value of the target variable instead of on its current value. For example, in trial #4 in Table 1 (a += 4), instead of considering a = 14 and responding 18, the participant would consider a = 6 and respond 10. *Relevant trials:* this error can occur only after the variable was updated at least once, i.e., not in the variable's first update in the code snippet.

Failed to shift: the calculation was not done on the target variable but on the variable that appeared in the previous trial – i.e., the participant did not shift to this trial's variable. For example, in trial 2 (b -= 4), instead of considering b = 12 and responding 8, the participant would consider a = 14 and respond 10. *Relevant trials:* switch trials in which the current-trial variable and the previous-trial variable have different values.

Failed to update & shift: the calculation was done on the previous value of the previous variable. For example, in trial 2, instead of considering b = 12, the participant would consider a = 6 and respond 2. In other words, the participant did not shift from a to b, and also erroneously used a's pre-trial value. *Relevant trials:* switch trials that follow a trial with correct response. If the response of the previous trial was incorrect, the previous trial's pre-trial value is ambiguous.

Incorrect shift: the calculation was done neither on the target variable nor on the variable from the previous trial, but on the third variable. In other words, the participant did perform a variable-shift relative to the previous trial, but shifted to the incorrect variable. For example, in trial 4 (a += 4), the participant would neither shift to a = 14 nor stay with c = 12, but would switch to b = 8 and respond 12. *Relevant trials:* switch trials in 3-variable code snippets.

Unnecessary shift: in a no-switch trial, the calculation was not done on the target variable but on a different variable. In other words, the participant performed a variable-shift when this was not needed. For example, in trial 5 (a -= 3), instead of considering a = 18 and responding 15, the participant would shift to one of the other variables, e.g., to c = 12, and respond 9. *Relevant trials:* no-switch trials. In 3-variable code snippets, in which there were two possible unnecessary shifts, a trial was deemed irrelevant if either of these shifts predicted a response outside the 1-20 range.

2.1.4.2.2 Calculation errors

+1: the response was larger than the expected response by 1. For example, in trial 3, instead of responding 12, the participant would respond 13.

-1: the response was smaller than the expected response by 1. For example, in trial 3, instead of responding 12, the participant would respond 11.

Two additional error types, +10 and -10, were defined similarly.

Incorrect operator: using the incorrect operator, i.e., addition instead of subtraction or vice versa. For example, in trial 2 (b -= 4), in which the pre-trial value is b = 12, instead of subtracting 4 and responding 8, the participant would add 4 and respond 16.

Forgotten carry: the calculation required decade crossing and the participant forgot to apply the carry (or borrow) procedure, i.e., to add/subtract 1 to/from the decade digit. As a result, the

response is smaller than the expected response by 10 for additions, and larger by 10 for subtractions. For example, in trial 3 (c += 9), in which the pre-trial value is c = 3, instead of responding 12, the participant would respond 2. *Relevant trials:* trials in which the calculation involves a decade crossing.

2.1.4.2.3 Other errors

No calculation: the response was the variable's pre-trial value, without performing any calculation. For example, in trial 1, in which the pre-trial value is a = 6, the participant would respond 6.

Double calculation: the addition or subtraction was applied twice. For example, in trial 5 (a -= 3), in which the pre-trial value is a = 18, instead of subtracting 3 and responding 15, the participant would subtract 3 twice, and respond 12.

11 instead of 9: for trials in which the right-hand-side operand is 9, considering 11 instead. For example, in trial 3 (c += 9), in which the pre-trial value is c = 3, instead of adding 9 and responding 12, the participant would add 11 and respond 14. The idea behind this error is that when the right-hand-side operand is 9, the calculation may be done in two steps: instead of computing "+ 9" the participant may compute "+ 10 - 1", and instead of computing "- 9" the participant may compute "- 10 + 1". The error type reflects a confusion in the flow of this calculation algorithm – executing "+ 10 + 1" instead of "+ 10 - 1", or "- 10 - 1" instead of "- 10 + 1". We classified this error as "others" and not as "calculation" because it reflects a mistake in the algorithm rather than in the simple math, and algorithm-execution errors may have various reasons, including malfunctions in working memory or in other runtime cognitive processes (Semenza et al., 1997; Zviran-Ginat, 2022). *Relevant trials:* trials in which the right-hand-side operand is 9.

2.1.4.3.*Statistical analysis*

We compared the rate of errors of each type to the chance level using Fisher's exact test. The chance probability for each error type is 1/19 – there are 19 possible erroneous answers in each trial, and each error predicts precisely one response. The only exception to this was the error "unnecessary shift": for 2-variable code snippets this error predicts one possible response, leading to a chance probability of 1/19 as explained above, but for 3-variable code snippets it predicts 2 possible responses, leading to a chance probability of 2/19. Thus, for this error type, we analyzed separately the code snippets with 2 variables and the code snippets with 3 variables.

2.1.5. Statistical analysis of factors affecting error rates and reaction times

The error rates were analyzed using logistic linear mixed models (LLMM) with the participant as a random factor. The fixed factors are described below for each analysis; the important factors were Switch (yes/no) and the Number of Variables in the code snippet (2 or 3). The reaction times of correct trials were analyzed with similarly-structured linear mixed models (LMM). To test the significance of a particular factor, we used a likelihood ratio test that compared the LMM or LLMM to an (L)LMM that was identical except it did not include the factor in question. We report the 1-tailed p-value of these comparisons, and the factor's coefficient in the full model (denoted Δ, because the predictors we used were binary, so the coefficient is close to the difference between the means of the two conditions).

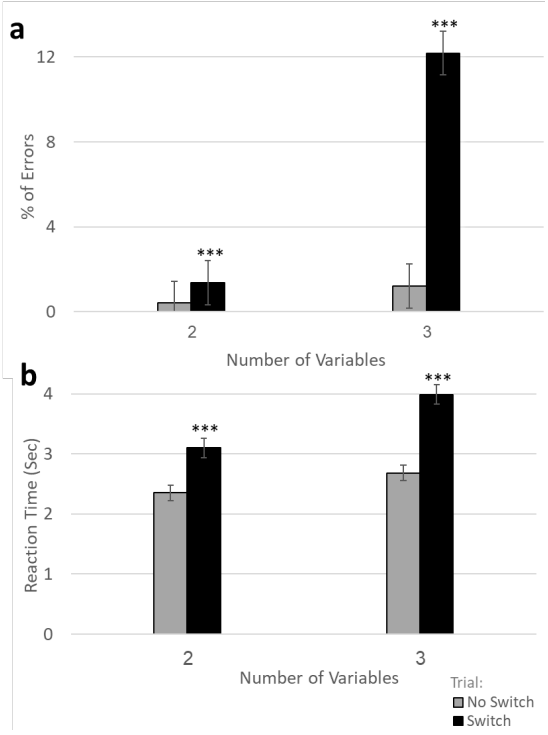
2.2. Results

2.2.1. General performance

We excluded 2 participants whose error rate was higher than 38.4% (this outlier threshold was defined as the third quartile plus 150% the interquartile range). After this exclusion, 30 participants remained. For these, 4.7% of the trials had ambiguous accuracy and were excluded from all analyses. The average error rate in the remaining trials was 14.3% (SD = 5.9%), and the average reaction time of the correct trials was 3,263 ms (SD = 840 ms).

The error rates and the reaction times were higher in the 3-variables code snippets than in the 2-variables code snippets (Fig. 1). To examine this difference statistically, we entered the accuracy (correct/incorrect) of each trial as the dependent variable into a logistic linear mixed model (LLMM) with the participant as the random factor and the Number of Variables (2, 3) as a single within-participant factor. The effect of Number of Variables was significant ($\chi^2 = 53.8$, $p < .001$, odds ratio = 1.61). Similar results were obtained when we used a regular (not logistic) linear mixed model

Figure 1. Error rates (a) and reaction times (b) for each trial, plotted separately for code snippets with 2 or 3 variables, and for trials with or without variable-switch. The harder conditions were the code snippets with 3 variables, and the switch trials. In these conditions, the error rates were higher and the reaction times were longer.



(LMM) on the reaction times of the correct trials (χ^2 = 138.2, p < .001, Δ = 300 ms). To conclude, remembering more variables is harder.

2.2.2. Switch trials are harder

The main question was whether the switch cost, which was observed in other WM-demanding tasks (Oberauer, 2002), exists also here, in a programming task. In other words, whether performance in the "switch trials", in which there was a switch from one variable to another relative to the previous trial, would be poorer than in no-switch trials. In this and subsequent analyses, we ignored the first trial in each code snippet, for which the switch / no-switch status is undefined because there is no preceding trial.

As predicted, there were more errors and longer reaction times in switch trials than in no-switch trials (Fig. 1). To test this effect statistically, we entered the accuracy (correct / incorrect) of each trial as the dependent variable into an LLMM, with the participant as the random factor and with 2 within-participant factors: Number of Variables (2, 3) and Switch (yes/no). The effect of Switch was significant (χ^2 = 863.6, p < .001, odds ratio = 2.7), even when we tested separately the 2-variable code snippets (without the Number of Variables factor, χ^2 = 35.8, p < .001, odds ratio = 1.61) and the 3-variable code snippets (χ^2 = 865.1, p < .001, odds ratio = 3.04). The effect of Switch was significant also in a similar LMM on the reaction time of the correct trials (χ^2 = 653.7, p < .001, Δ = 570 ms), also when testing separately the 2-variable code snippets (χ^2 = 110.5, p < .001, Δ = 360 ms) and the 3-variable code snippets (χ^2= 551.6, p < .001, Δ = 650 ms).

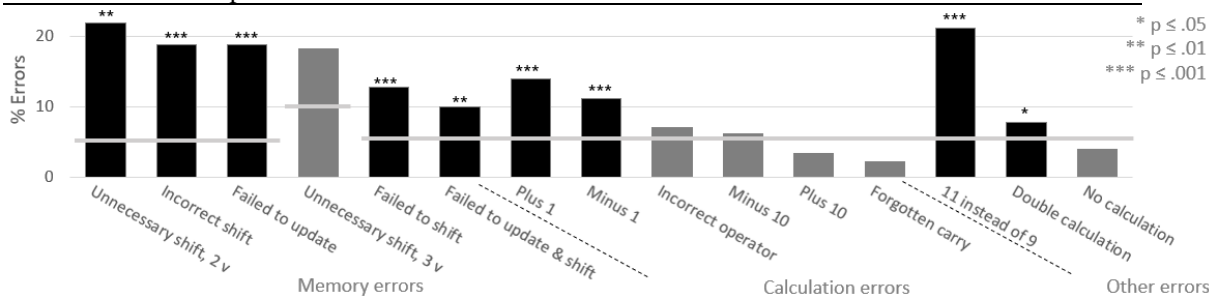
These findings show a clear switch cost: replicating Oberauer (2002), switch trials were harder than no-switch trials, also in our programming task.

2.2.3. Error analysis

Our interpretation of the results is that the process supporting the variable switch, presumably shifting the Focus of Attention in working memory, created cognitive load. To further show that the difficulty in the task was indeed specifically due to shifting the Focus of Attention (FOA), we analyzed the types of errors made by the participants. If the difficulty was caused by shifting the FOA, we expect the errors to be typical to shift confusions (for example: shifting to an incorrect variable), whereas other types of errors – for example, calculation errors – should not be as frequent.

2.2.3.1. *Results*

Figure 2. The rate of errors of each type. The thick line shows the chance level for each error type. Asterisks denote the comparison to chance level using Fisher's exact test. The working-memory-related errors were the most prevalent ones.

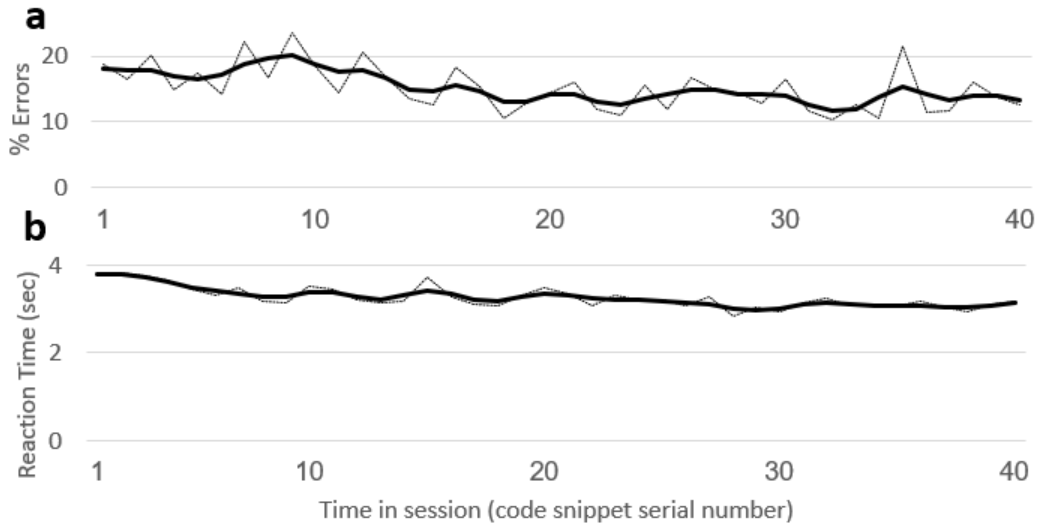


Comparing the error rate in each type to the chance level using a Fisher's exact test, the error rate was significantly higher than chance for 9 of the 15 analyzed error types. These 9 error types reflect the participants' real confusions. Importantly, 5 of the 6 memory error types were significantly higher than chance, but only 2 of the 6 calculation error types (+1, -1), and even these two were not the most prevalent error types. In addition, when considering only the error types that were significantly above chance level, 54.4% of all incorrect trials were classifiable as memory errors, whereas only 36.6% were classifiable as calculation errors. These findings provide additional evidence to the idea that most errors indeed originated in memory confusions and not in the calculation process.

An unexpected finding was that the second most common error type was "11 instead of 9" – an error type that we did not classify as a memory error. This error is unique because it indicates a confusion in the execution of the algorithm, i.e., in applying the correct sequence of mathematical operations. However, in retrospect, these errors too may be related to WM. A possible explanation for the high rate of such errors is that working memory is crucial to follow the correct sequence of operations in a mental algorithm (Brunyé et al., 2006; Cragg et al., 2017; Hubber et al., 2014), and these errors too are affected by the working-memory load in this task. Interestingly, this algorithmic confusion was unique to the case of 9 and 11 (10 ± 1, 21.2% errors); the error rate was lower for parallel number pairs like 8 and 12 (10 ± 2, 2.6% errors), 7 and 13 (10 ± 3, 10.0%), 6 and 14 (10 ± 4, 9.3%), and others (< 6% errors for each). This may be because people tend to use the "+10-1" strategy for 9 more than for other numbers, or because the right-hand-side operand 9, but not operands with smaller values, leads to precisely the same operation (either +1 or -1) in the decade and the unit positions.

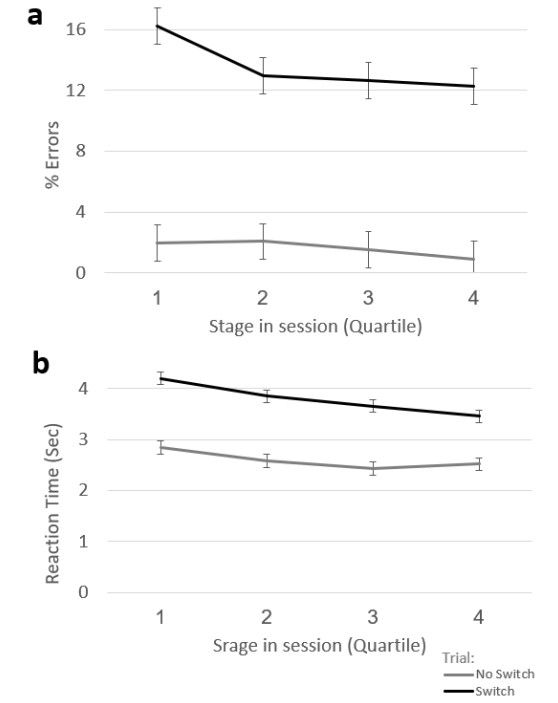
2.2.4. Selective improvement in variable shifting

Figure 3. Average Error rate (a) and reaction times (b) in each code snippet throughout the session. The thick line shows the data after Gaussian smoothing (σ= 1).



The performance improved throughout the session – both the error rates and the reaction times (Fig. 3). That is, there was a learning effect. There may have been a fatigue effect alongside, but if such an effect indeed existed – it was smaller.

Figure 4. Error rates (a) and reaction times (b) in each quartile of the experiment session, plotted separately for switch and no-switch trials.



What precisely was the ability that improved during the session? The trivial explanation is task-specific improvement: presumably, when executing any task for sufficiently long, the performance improves. A more interesting possibility, however, is a specific improvement in working-memory. Such an improvement may affect only the switch trials because these trials require shifting information in working memory, whereas the no-switch trials do not require such a shift.

Supporting this possibility, the decrease in error rate and reaction time during the session seemed larger in switch trials than in no-switch trials (Fig. 4). To test this effect statistically, we entered the accuracy (correct/incorrect) of each trial as the dependent variable into a LLMM with the participant as the random factor, and with Code snippet (1-40, numeric) and Switch (yes/no) as within-subject

factors. The Code snippet x Switch the interaction was significant (χ^2 = 70.3, p < .005, odds ratio = 3.57). Similar results were obtained when we used a LMM on the reaction times of the correct trials (χ^2 = 64.5, p < .01, Δ = 600 ms).

The above analysis of reaction times could be criticized on the grounds that the overall error rates were extremely different in the two conditions (switch, no-switch). Given such a difference, the analysis' null hypothesis, namely that the improvement in the two conditions would be equal numerically – could be questioned. To address this criticism, we normalized the reaction times to a common scale by dividing each trial's reaction time by the condition's (switch / no-switch) grand average. We entered the normalized RT as the dependent variable in a LMM with Code snippet (1-40, numeric), Switch (yes/no), and their interaction as within-subject factors. The interaction effect was still significant (χ^2 = 3.65, p = .006, Δ = 50 ms), confirming that the selective improvement in switch trials was not a confound of the overall difference in RT between switch and no-switch trials.

This selective improvement in switch trials, which was larger than in the no-switch trials, indicates that the origin of the improvement was in a process involved uniquely the switch situation – presumably, the refocusing on a different variable, i.e., shifting the Focus of Attention in working memory.

2.2.5. The effect of inactivity duration

Our main hypotheses concerned the variables stored in the Focus of Attention, i.e., the variable on which the participant performed the calculation in each trial (hereby, the "active variable"). However, our data can inform also about the WM regions with lower accessibility levels - the Region of Direct Access and the Activated part of Long-Term-Memory – which presumably store the currently **inactive** variables, namely the 1 or 2 variables that do not appear in the present trial. We asked whether, in 3-variable code snippets, the two currently-inactive variables were stored in the same accessibility level, and if not – why.

To examine this, we classified the trials according to the variable's inactivity duration – the number of consecutive trials during which the trial's variable was inactive before the current trial. For example, the inactivity duration of no-switch trials is 0. We compared the error rate between trials with inactivity duration = 2 (i.e., the variable was last used 3 trials ago) and trials with inactivity duration = 1. We included only 3-variables code snippets in this analysis, because in 2-variable code snippets the maximal inactivity duration was 1. We excluded the first use of each variable, for which the inactivity duration is undefined.

The error rate in trials with inactivity duration = 2 (35.2%) was higher than in trials with inactivity duration = 1 (23.8%). To test this effect statistically, we entered accuracy (correct/incorrect) of each trial as the dependent variable into a LLMM, with the participant as the random factor, and with Inactivity Duration (1, 2) as a single within-participant factor. The effect was significant (χ^2 = 19.6, p < .001, odds ratio = 1.46). Thus, the accessibility level of the currently inactive variables is not fixed, but it depends on a variable's inactivity duration: higher inactivity duration entails more errors once the participant needs to bring it to the FOA.

2.3. Discussion

Our findings confirmed the main prediction: switch trials were harder than no-switch trials, both in terms of error rates and of reaction times. This switch-cost replicates the results of Oberauer (2002), now in the context of a programming task, and reaffirms his conclusion that shifting the Focus of Attention (FOA) from one information item to another has a cost. We also extend Oberauer's findings in two ways, which provide further support to the conclusion: first, we observed the switch cost not only in the reaction times but also in the error rates. Second, we analyzed the error types and we showed that the most common errors were not calculation errors but memory-related confusions, in particular several kinds of incorrect variable-switching, supporting the idea that these errors were related to the FOA-switch mechanisms.

We also observed two additional findings, allowing for two additional conclusions. First, the currently inactive variables were not all stored with the same accessibility level. Rather, the accessibility level, as reflected in the error rates, was affected by the variable's "inactivity duration", the number of trials elapsed since a variable was last used, such that longer inactivity duration entailed more errors. This finding illuminates on the interplay between the Region of Direct Access (RDA) and the Activated part of Long-Term Memory (ALTM), because it refutes the possibility that all inactive variables were stored in the RDA, which is supposed to have a fixed accessibility level (Oberauer, 2002, 2020). We return to this point in the General Discussion.

The second finding was an improvement throughout the session. Critically, this improvement was larger in switch trials than no-switch trials. We hypothesize that this improvement originates in a selective training of the variable-shift mechanism in the working memory. Experiment 2 examined this issue further.

3. Experiment 2

Experiment 1 showed a selective improvement in the switch trials. Our interpretation of the task according to Oberauer's (2002) working-memory model can readily explain this pattern as an improvement in the FOA switch mechanism. However, because we did not predict this specific finding in advance, Experiment 2 was designed to replicate it in slightly different, and more controlled, experiment setting.

The experiment included 30 code snippets. To examine improvement, we compared the first 5 code snippets in the session (hereby, 'pre-session' code snippets) to the last 5 code snippets ('post-session' code snippets; the 20 code snippets in between are the 'training' code snippets). The participants also performed, two days later, a second session with 10 'follow-up' code snippets, to examine whether the improvement would persist in the long term. The pre-session, post-session, and follow-up code snippets were balanced in several parameters, detailed below, to ensure they had comparable degree of difficulty.

There were a few additional differences between Experiment 2 and Experiment 1. First, the range of valid variable values was not 1-20 but 0-10. The reason was to avoid cross-decade calculations, which may entail additional working memory operations needed to implement the carry procedure (Nir, 2023). Second, unlike Experiment 1, in which the participants had to update the variable value in their memory after each calculation, here the calculation trials did not change the variable value, i.e., the variables retained their initial value throughout the code snippet. To encourage the participants to conform to this instruction and return, after each calculation, to the variable's pre-trial value, each calculation trial was followed by a "query" trial in which the participant had to type the pre-trial value. For example, a trial such as "x+4" was followed by a "x?" trial.

Unlike Experiment 1, here the participants were not instructed to stick to an erroneous response they gave and to keep using this value in the next trials. Correspondingly, again unlike Experiment 1, the analysis of errors considered each variable's correct value and ignored errors made on previous trials.

3.1. Method

3.1.1. Participants

The participants were 30 adults (aged 18-35) with normal or corrected-to-normal vision and no reported cognitive disorders.

3.1.2. Stimuli

Types of code snippets. The first session included 30 code snippets. Unknown to the participants, they were divided into pre-session code snippets (first 5), training code snippets (next 20), and post-session code snippets (last 5). The order of code snippets was randomized for each participant, but a particular code snippet being a pre-session, post-session, or training snippet was the same for all participants. The second session included 10 code snippets, which were identical to the first session's pre-session and post-session code snippets, again in random order.

Variables. Each code snippet included 3-4 variables, whose value was in the range 0-10. The variable names were g-n-x-t, j-b-f-u, p-m-e-k, z-r-h-s in different code snippets, counterbalanced. The pre-session and post-session code snippet included 4 variables each. We made sure the pre-session code snippets were similar to the post-session code snippets in terms of initial variable values (pre-session: M=5, SD=1.7; post-session: M=4.7, SD=2), the calculation's right-hand-side operand (pre-session: M=0.4, SD=3.5; post-session: M=0.3, SD=3.4), and the calculation operators (additions in pre-session: 57%; post-session: 55%). The 20 training code snippets had either 3 variables (5 code snippets) or 4 variables. The 10 code snippets of session 2 were identical with the pre-session and the post-session code snippets, only in different (random) order.

Code snippet structure. Each code snippet included 25 trials. The first trial initialized the variables to different values in the range 2-8. It was followed by 12 pairs of trials, each pair referring to one variable (3 pairs per variable in the 4-variable code snippets, and 4 pairs per variable in the 3-variable code snippets). In each pair, the first trial was a *calculation* trial, asking to add or subtract a number to one of the variables (e.g., "a + 10"); and the second (*query* trial) asked to type the pre-calculation value of the same variable ("a?"). The purpose of the query trial was to ensure that the participant would mentally return to the variable's pre-trial value. Because the calculation trials did not change the variable value, they were presented without the '=' operator (e.g., "a+4").

Experimental conditions. We manipulated the congruency between each calculation trial and the preceding query trial, i.e., whether both referred to the same variable (no-switch trial, 50% of the

calculation trials in each code snippet) or not (switch trial). The query trials did not have this manipulation – they always referred to the same variable as the preceding calculation trial. The congruency of the first calculation trial in each code snippet was undefined, so this trial was excluded from the congruency analysis.

3.1.3. Procedure

The procedure was as in Experiment 1, with the participant responding to each trial by typing the variable value (except the first, variable-initialization trial). However, because the variable values were now in the range 0-10, the response keys were 1-9, and the keys to their left (“`”) and right (“0”) for 0 and 10, respectively. As in Experiment 1, “tab” and “[” were used to respond ‘< 0’ and ‘> 10’, respectively (but this was never the correct response).

3.1.4. Error coding

We classified the errors into types as in Experiment 1, but here we used only 8 error types.

Memory errors: Failed to Shift, Incorrect Shift, Unnecessary Shift. The error types "failed to update" and "failed to shift & update" could not occur here, because there was no variable-update in this experiment. *Relevant trials:* as defined in Experiment 1; additionally, trials following an incorrect query trial were deemed irrelevant.

Calculation errors: +1, -1, Incorrect Operator. The other types of calculation errors could not occur because they concern cross-decade calculations, which did not occur in this experiment.

Other errors: No Calculation, Double Calculation. The error "11 instead of 9" could not occur here, because the value range was 0-10.

We compared the rate of errors of each type to chance level using Fisher's Exact test. The chance probability for each error type is 1/10 – there are 10 possible erroneous answers in each trial, and each error predicts precisely one response. There were two exceptions to this. First, the chance level of the "incorrect shift" error type depended on the code snippet: in 3-variable code snippets this error predicts one possible error, leading to a chance probability of 1/10, but in 4-variable code snippets it predicts 2 possible responses, leading to a chance probability of 2/10. The second exception was the error "unnecessary shift": for 3-variable code snippets this error predicts 2 possible responses, leading to a chance probability of 2/10, and for 4-variable code snippets it predicts 3 possible responses, leading to a chance probability of 3/10. Thus, for these error types, we analyzed separately the code snippets with 3 variables and the code snippets with 4 variables.

3.2. Results

The average error rate in the calculation trials was 22.1% (SD = 15.9%) and the average reaction time of the correct responses was 2,550 ms (SD = 521 ms). The average error rate of the query trials was 21% (SD = 16%) and the average reaction time of the correct responses was 665 ms (SD = 318 ms).

3.2.1. Selective improvement in variable shift?

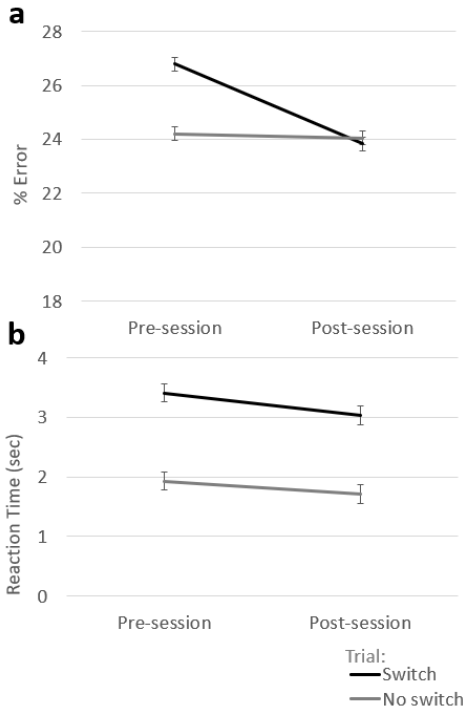


Figure 5. Error rate (a) and reaction times (b) in the pre-session and the post-session, plotted separately for trials with and without variable-switch.

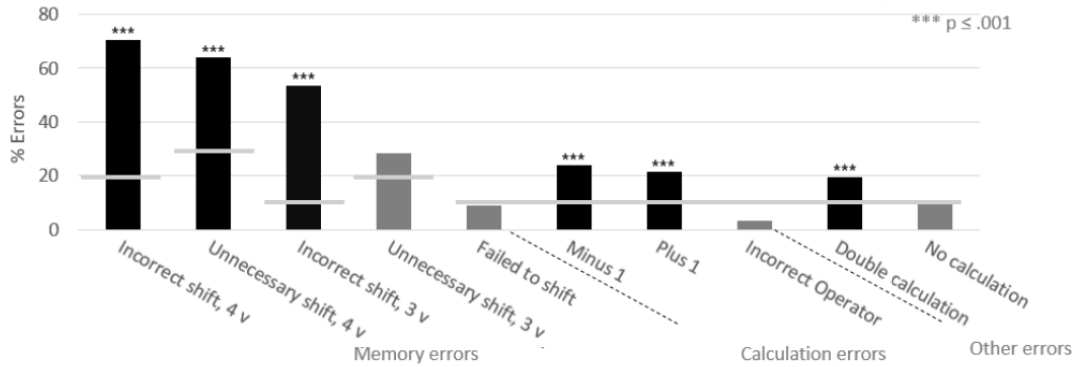
To examine whether a learning effect occurred here similar to Experiment 1, we compared the error rates and reaction times between the pre-session and the post-session code snippets (Fig. 5). Error rate decreased from the pre-session to the post-session trials by 2.8% in the switch trials but by only 0.4% in the no-switch trials. Reaction times decreased by 380 ms in the switch trials but by only 220 ms in the no-switch trials. Thus, the improvement in the switch trials was numerically larger than the no-switch trials. However, this effect did not reach significance. We ran an LLMM on the per-trial accuracy (correct/incorrect) with the participant as the random factor, and with Time (pre-session/post-session), Switch (yes/no), and their interaction as within-participant factors. The Time effect was not significant ($\chi^2 = 1.25$, $p = .26$), nor was the Time × Switch interaction term ($\chi^2 = 1.01$, $p = .31$). The Time × Switch interaction was not significant also in a similar LMM on the reaction times of the correct trials ($\chi^2 = 1.45$, $p = .22$), although the Time effect was significant ($\chi^2 = 21.6$, $p < .001$, $\Delta = 144$ ms).

In short, contrary to Experiment 1, we did not observe a significantly larger improvement in switch trials than in no-switch trials.

3.2.2. Error types

As in Experiment 1, we hypothesized that the difficulty in the task originated in the mechanism supporting variable-switches, in particular, shifting the FOA, and we predicted that this would be reflected in shift-related memory errors being more frequent than other types of errors.

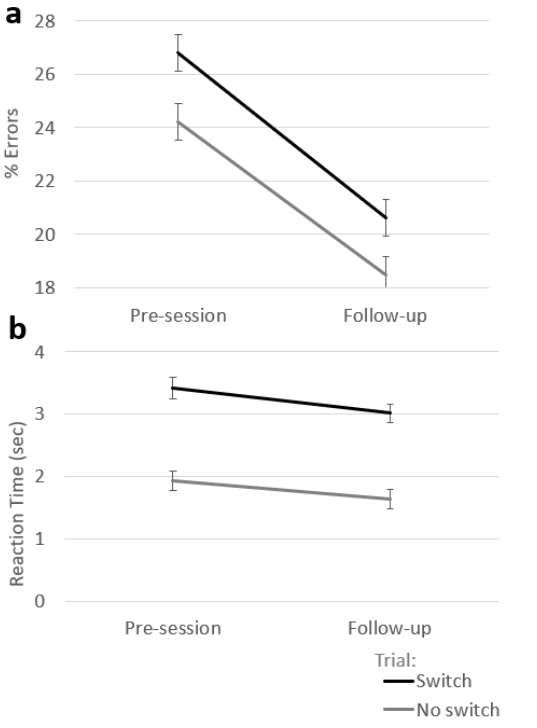
Figure 6. The rate of errors of each type. The thick line shows the chance level of each error type. Asterisks denote the comparison to chance level using Fisher's exact test. The working-memory-related errors were the most prevalent ones.



This was indeed the case: the number of errors was significantly higher than chance for 6 of the 10 analyzed error types, and critically, the most prevalent errors were the memory errors. These findings replicate Experiment 1, and support the idea that the source of most errors is in the FOA-shifting mechanisms and not in the calculation process.

3.2.3. Long-term improvement

Figure 7. Error rate (a) and reaction times (b) in the pre-session and the follow-up, plotted separately for trials with and without variable-switch.



We next examined whether the experiment created improvement that persisted even two days later; and whether such putative improvement was larger in the switch trials (Fig. 7). Comparing the pre-session code snippets (in session 1) to the follow-up code snippets (session 2), the error rate decreased by 6.2% in switch trials and by 5.7% in no-switch trials; and the reaction time decreased by 400 ms in switch trials and by 300 ms in no-switch trials. An LLMM on the accuracy (correct/incorrect) of each trial, with the participant as the random factor and with Time (pre-session/follow-up), Switch (yes/no), and their interactions as within-subject factors, showed a significant improvement (Time effect: $\chi^2 = 0.001$, $p < .001$, odds ratio = 1.22), however, the Time × Switch interaction was not significant ($\chi^2 = 0.01$, p = .97). Similar results were obtained when we used a LMM on the reaction times of the correct trials (Time effect: $\chi^2 = 46.8$, $p < .001$, $\Delta = 176$ ms; Time × Switch interaction: $\chi^2 = 1.51$, p = .23).

Thus, there was a long-term improvament in the task performance, however, improvement was not larger in the switch trials than in the no-switch trials.

3.3. Discussion

Experiment 2 failed to replicate Experiment 1: we did not find larger improvement in the switch trials than in the no-switch trials – neither at the end of the session nor two days later. Thus, the experiment did not confirm the main prediction of a selective training of the variable-shift mechanism in the working memory. Numerically, the results were in the predicted direction, with numerical difference between the improvement in switch and no-switch trials, both in term of error rate and reaction times, however, these differences were not significant. Similarly, in the follow-up testing 2 days later, we observed an improvement relative to the pre-session testing but no selective improvement specifically in the switch trials compared to the no-switch trials. Still, we did observe an overall improvement from the pre-session test to the test conducted 2 days later; and we did see more memory errors than other errors, supporting the claim that most errors resulted from the FOA-shift confusions.

Below, in the General Discussion, we propose possible explanations for the different results between Experiment 1 and Experiment 2.

4. General Discussion

This study examined the process of shifting the Focus of Attention (FOA) in working memory in a simple programming task, in which the participants tracked in their minds the values of several variables. In line with the idea that shifting the FOA requires cognitive effort, we observed poorer performance (switch cost) in trials that required shifting the Focus of Attention. This switch cost replicates the findings of Oberauer (2002) in the domain of programming; and our analysis of error types, which showed a prevalence of shift-related errors, extends his finding and further support the idea of an effortful FOA-shifting. Moreover, our data suggest a possible improvement, during the session, of the FOA shifting ability. We now elaborate on each of these issues

4.1. Shifting the Focus of Attention

Shifting the Focus of Attention (FOA) from one information item to another in WM requires some degree of cognitive effort. In our experiments, in which each trial focused on a single variable, this effort was reflected in the switch cost: switch trials, which involved a variable different from that of the previous trial, were harder (slower, more errors) than no-switch trials, which involved the same variable as in the previous trial. Presumably, in the no-switch case, the FOA was already set on the correct variable because the participant focused on it in the previous trial, but switch trials required shifting the FOA to a different variable.

This switch cost replicates the findings of Oberauer (2002), who showed a similar switch cost in a task in which the updated values were not presented as variables in a code snippet but as values associated with different "slots" in different frames on the screen, or different colors (Oberauer, 2002, 2005). Our data also extends Oberauer's findings in two ways:

1. **Error rate.** Oberauer (2002) showed a switch cost in the reaction times; here, we replicated this effect but we also extended it to error rates: the error rate of switch trials was larger than the error rate of no-switch trials. This is important for two reasons. First, it shows beyond any doubt that the difference in rection time was not a matter of speed-accuracy tradeoff. Second, it demonstrates that in a real-world context, the extra effort imposed by the FOA switching causes not just slowness, but even mistakes.
2. **Error classification.** On top of analyzing error rates and reaction times, we also analyzed the types of errors made by the participant – a method not used in previous studies. Most of the detectable error types were related to variable-switching: switching to an incorrect variable, switching to another variable when this was not needed, or failing to switch when this was

needed. These switches are precisely the activity driven by the Focus-of-Attention shifting process. Correspondingly, the switch-related errors presumably reflect a malfunction of this process – e.g., the selection of an incorrect information item among the information items competing in the RDA (crosstalk; Oberauer, 2002) or a confusion in carrying out the FOA shift.

This finding strongly supports the idea that switch trials were hard because of the need to shift the Focus of Attention.

Another finding was that the error rates and reaction times were higher in 3-variable code snippets than in 2-variable code snippets. This finding is not surprising and could have many explanations. One possible explanation is that when the RDA includes more candidate information items from which one will be selected to the FOA, the selection process becomes harder, requires more time, and slows the completion of the cognitive operation applied on the selected information item (Oberauer, 2002).

4.2. Training the FOA-shift

4.2.1. Selective improvement in switch trials

Another finding, which was reported here for the first time, is that the participants' performance improved during the session, both in the error rates and in the reaction times. Improvement in a task is not surprising, but importantly, the improvement was larger in the switch trials than in the no-switch trials. This pattern was observed even when we controlled for the different overall degrees of difficulty in the two conditions. The selective improvement in switch trials suggests that the improvement originated in a mechanism that specifically operated in the switch trials – the FOA shifting.

The possibility of improving WM skills is interesting and was examined in several studies (Melby-Lervåg & Hulme, 2013). The motivation is clear: WM influences on many cognitive processes, and WM capacity has been found linked to many skills (Melby-Lervåg & Hulme, 2013), so naturally, training and improving WM raises much interest. If improving the WM is possible, it could help having better memory skills, better ability to learn new things easily, to develop cognitively, etc. Indeed, some researchers have already claimed that the working memory can be improved – for example, by extending the capacity of the FOA from one information item to four information items (Verhaeghen et al., 2004). However, a selective improvement of FOA shifting was not reported previously.

4.2.2. Failure to replicate the improvement in Experiment 2

We observed the aforementioned improvement in Experiment 1, in which it was an unexpected finding. However, Experiment 2 failed to replicate the selective improvement in switch trials, neither during the session nor in the long term, after 2 days. Although Experiment 2 showed a larger decrease numerically in the error rate and reaction time in switch trials than in no-switch trials, this effect was not significant.

Our data cannot point unambiguously at the reason for this discrepancy between the two experiments. However, there were few methodological differences between Experiment 1 and Experiment 2, which may explain the different results:

1. **Update versus no-update.** In Experiment 1, after performing a calculation based on a certain variable, the participants were to update the variable in their WM to the new value and discard the pre-trial value. In Experiment 2, the participants still performed a calculation on one variable in each trial, but they were instructed not to update the variable value in their WM, but to stick to the pre-trial value. To ensure they do this, we even added a query trial after each calculation trial, in which the participant was asked about the pre-trial value.

   These different instructions entail that slightly different cognitive processes were invoked in each experiment. First, Experiment 1, but not Experiment 2, involved updating of the memorized values in WM on each trial. Second, although both experiments required removal of information from working memory at the end of each trial, the information removed was different in each experiment: the pre-trial value in Experiment 1, and the calculation result in Experiment 2. It could be that the selective improvement observed in Experiment 1 was in one of the processes that operated only in this experiment – e.g., a process related to the updating of WM.

2. **Query trials.** These trials, which followed each calculation trial and aimed to refocus the participant on the pre-trial value, occurred in Experiment 2 but not in Experiment 1. The existence of query trials may have affected the FOA-shift process. The query trial, by asking the participants to "return" to the pre-trial value, may help the participants refresh their memory and "boost" the representation of items stored in WM, consequently improving recall (Vergauwe et al., 2023). This memory boost may affect the trial's overall difficulty, the amount of challenge (and training) imposed by the consecutive switch, and it may even have different effects on switch and no-switch trials.

3. **Calculation range**. In Experiment 1, the range of correct responses was 1-20, whereas in Experiment 2, the range was 0-10. The cross-decade calculations, which occurred only in Experiment 1, may have created additional working memory load and additional shifts in the FOA (Nir, 2023), and in turn – more WM training. Moreover, the extra load imposes by the cross-decade calculations may have also encouraged the participants in Experiment 1 to move information between the RDA and ALTM (if the RDA's capacity was insufficient) – another kind of WM shifts.
4. **Population**. The participants in the two experiments were drawn from different populations: in Experiment 1, they had minimal knowledge in programming, whereas in Experiment 2 they did not. These differences may have perhaps affected the participants' learning potential. For example, it may be that the programming-knowers had higher memory skills, and that such individuals gain more from cognitive training. A similar "the rich get richer" effect in training is known from other domains (Stanovich, 2009).

Future studies may use the differences mentioned here, and manipulate each of them separately, to identify which are the critical factors for effective training of FOA-shifting.

4.2.3. Which type of training is most efficient?

One idea, which was not tested systematically in this study, is that the best way to improve the FOA shifting process, and consequently the performance in switch trials, is to train the participants specifically on these trials. We did not run a full experiment to test this idea directly, however, several pilots we ran failed to obtain this effect. We manipulated the training type (intensive, with many switch trials, or non-intensive, with few switch trials), but contrary to our prediction, the intensive training was no better than the non-intensive training in selectively improving in the switch trials. Our impression was that when a training session included many hard code snippets (as was the case in the intensive training), the training was too difficult and hence ineffective. This idea accords with the flow of creativity theory (Nakamura & Csikszentmihalyi, 2009), a crux of which is the balance between challenge and skill: when a person performs a task in which both the person's skill and the challenge imposed by the task are above average, and they are also balanced one versus the other (relative to that person's abilities), the person can operate at full capacity and fully absorb the activity he does. In contrast, if the challenge and skill are not balanced, e.g., the challenge is too high relative to the skill (as may have been the case in our pilots), the person become vigilant and anxious, and performance is poorer.

Following this series of pilots we assume that for the training to be effective, it must be well-balanced at least in two ways. First, in terms of difficulty – not too easy, not too hard. Such balancing would have to consider, for the very least, the two main factors of difficulty we observed here: the number of variables and the proportion of switch trials. Second, the training must also be in the appropriate length – long enough to get sufficient training, but not too long. It is possible that our experiments were too short; indeed, other working memory improvement experiments used longer training (Melby-Lervåg & Hulme, 2013).

A related issue concerns not how effective the training is, but how good is our ability to measure it. In Experiment 2, unlike Experiment 1, if the participants made an error, they were allowed to self-correct it later. This instruction caused some ambiguity in the interpretation of results and may have partly masked our ability to observe any training effect. In retrospect, the Experiment 1 method seems better in this sense.

4.3. The effect of inactivity duration

We found the error rate of trials with inactivity duration = 2 (i.e., this variable was last used 3 trials ago) was higher than trials with inactivity duration = 1 (the variable was used 2 trials ago). This finding indicates that higher inactivity duration of the variables in the RDA entails more errors once needed to be brought to the FOA.

This finding illuminates on the interplay between the Region of Direct Access (RDA) and the Activated part of Long-Term Memory (ALTM). Oberauer (2002, 2020) made two claims concerning these regions: (a) There are no accidental or unintended 'drops' from the RDA to the ALTM – all information relevant to current goal remain in the RDA, subject to its capacity limit; and (b) all information items in the RDA have the same accessibility level. The finding of an inactivity duration effect can therefore have few interpretations.

One possibility is that some items are removed from the RDA because it is overloaded – e.g., the load imposed by the two inactive variables plus the load imposed by the calculations involved in cross-decade calculations. Under this view, the inactivity duration effect may reflect that on each trial, an item has a certain likelihood to be removed from the RDA, and the longer it remains inactive, this likelihood increases. If this is the case, the inactivity duration effect may be interesting methodologically – it may be used as a tool to examine which operations load on the RDA.

A second possibility is that different variables in the RDA have different levels of accessibility. This is interesting because it shows that some assumptions of Oberauer's model (2002) were incorrect.

A third possibility concerns the process via which information items are removed from the RDA (but remain in the ALTM). Presumably, by default information is retained in the RDA, and it exits the RDA only via active removal (Lewis-Peacock et al., 2018; Oberauer, 2020). It may be that there was an active removal, or decay of one of the variables from the RDA, although it was still relevant to the task, contrary to Oberauer's (2020) assumptions.

Another issue concerns the extent of inactivity duration. Here, we compared between variables with activity duration 2 and 1, because there were not enough trials in which the inactivity duration was longer. Future studies may use a design with longer inactivity durations, presumably in 3-variables code snippets, to examine if the effect is continuous even in longer inactivity durations.

The effect of inactivity duration was found in Experiment 1 but not in Experiment 2. The reason for this may be the different mental processes that the participants were asked to perform. For example, it may be that Experiment 2 imposed lower WM load, because there were no cross-decade additions, so there was no need to remove the inactive variables from the RDA. Another explanation could highlight that in Experiment 2 there was no update of the variables, and the process of removing the unnecessary value was different from Experiment 1. The presence of these updates, which intrinsically may involve some removal operation (Lewis-Peacock et al., 2018), may have increased the likelihood for unnecessary removals of information.

4.4. Investigating working memory in the context of a programming task

In both experiments, the task we used was a simple programming task. This is important, because it opens the door to examining the relevance of WM functions to programming, a relatively new field that has become an important skill in modern society.

To understand these relations, we must first understand which aspect of programming was examined here. This is not trivial, because "programming" is a very wide concept, which includes many activities and correspondingly, many cognitive aspects. Very broadly, programming can be divided into ***problem solving***, i.e., the challenges involved in understanding the problem at hand, breaking down the problem into smaller units, and designing the algorithm/s that will solve each unit (Fedorenko et al., 2019); and ***coding***, i.e., the "translation" of the algorithm into a specific programming language. Traditionally, most research on programming has focused on the problem-solving aspect of programming. However, programmers spend most of their time on program comprehension – reading and understanding the existing source code (Minelli et al.,

2015), which is more related to coding than to problem solving, and still little is known about these skills (Ivanova et al., 2020; Siegmund, 2016). The tasks we used here focused on the coding aspect.

Even coding, and specifically code comprehension (which was the task here), is not a unitary concept. Code comprehension has two different aspects: knowing/understanding what the source code means, i.e., "what it does"; and understanding what will happen once that code is executed. These two things are quite different, similar to how understanding the arithmetic algorithm of multi-digit addition is not the same as actually computing 45+89 in your head.

Existing studies on code comprehension used different methodological paradigms (Siegmund, 2016), but they almost invariably focused on the "knowing/understanding" aspect. In contrast, here we examined the participants' ability to execute code in their heads. We observed the important role of WM in this task; this is not surprising, and indeed WM plays a central role also in the execution of mathematical algorithms (Raghubar et al., 2010; Semenza et al., 1997).

Importantly, the present study "connects" computer programming and working memory in a more precise manner than in the past. Previous studies nicely showed that memory plays a role in understanding a code program (Siegmund, 2016). Here, we took these findings an additional step forward, and showed a highly specific working-memory mechanism that might underlie some of these correlations.

References

Baddeley, A. (1992). Working Memory. *Science*, *255*(5044), 556–559. https://doi.org/10.1126/science.1736359

Brunyé, T. T., Taylor, H. A., Rapp, D. N., & Spiro, A. B. (2006). Learning procedures: The role of working memory in multimedia learning experiences. *Applied Cognitive Psychology*, *20*(7), 917–940. https://doi.org/10.1002/acp.1236

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, *24*(1), 87–114. https://doi.org/10.1017/S0140525X01003922

Cowan, N. (2017). The many faces of working memory and short-term storage. *Psychonomic Bulletin & Review*, *24*(4), 1158–1170. https://doi.org/10.3758/s13423-016-1191-6

Cragg, L., Richardson, S., Hubber, P. J., Keeble, S., & Gilmore, C. (2017). When is working memory important for arithmetic? The impact of strategy and age. *PLOS ONE*, *12*(12), e0188693. https://doi.org/10.1371/journal.pone.0188693

Fedorenko, E., Ivanova, A., Dhamala, R., & Bers, M. U. (2019). The Language of Programming: A Cognitive Perspective. *Trends in Cognitive Sciences*, *23*(7), 525–528. https://doi.org/10.1016/j.tics.2019.04.010

Garavan, H. (1998). Serial attention within working memory. *Memory & Cognition*, *26*(2), 263–276. https://doi.org/10.3758/BF03201138

Hubber, P. J., Gilmore, C., & Cragg, L. (2014). The Roles of the Central Executive and Visuospatial Storage in Mental Arithmetic: A Comparison across Strategies. *Quarterly Journal of Experimental Psychology*, *67*(5), 936–954. https://doi.org/10.1080/17470218.2013.838590

Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'Reilly, U.-M., Bers, M. U., & Fedorenko, E. (2020). Comprehension of computer code relies primarily on domain-general executive brain regions. *eLife*, *9*, e58906. https://doi.org/10.7554/eLife.58906

Lewis-Peacock, J. A., Kessler, Y., & Oberauer, K. (2018). The removal of information from working memory. *Annals of the New York Academy of Sciences*, *1424*(1), 33–44. https://doi.org/10.1111/nyas.13714

McElree, B. (2001). Working memory and focal attention. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *27*(3), 817–835. https://doi.org/10.1037/0278-7393.27.3.817

Melby-Lervåg, M., & Hulme, C. (2013). Is working memory training effective? A meta-analytic review. *Developmental Psychology*, *49*(2), 270–291. https://doi.org/10.1037/a0028228

Minelli, R., Mocci, A., & Lanza, M. (2015). I Know What You Did Last Summer—An Investigation of How Developers Spend Their Time. *2015 IEEE 23rd International Conference on Program Comprehension*, 25–35. https://doi.org/10.1109/ICPC.2015.12

Nakamura, J., & Csikszentmihalyi, M. (2009). Flow Theory and Research. In S. J. Lopez & C. R. Snyder (Eds.), *The Oxford Handbook of Positive Psychology* (pp. 194–206). Oxford University Press. https://doi.org/10.1093/oxfordhb/9780195187243.013.0018

Nir, S. (2023). *How we perform carry operation, why is it so difficult, and what can be learned from it about algorithms?* Unpublished MA dissertation, Tel Aviv University.

Oberauer, K. (2002). Access to information in working memory: Exploring the focus of attention. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *28*(3), 411–421. https://doi.org/10.1037/0278-7393.28.3.411

Oberauer, K. (2005). Control of the Contents of Working Memory—A Comparison of Two Paradigms and Two Age Groups. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *31*(4), 714–728. https://doi.org/10.1037/0278-7393.31.4.714

Oberauer, K. (2020). Towards a Theory of Working Memory: From Metaphors to Mechanisms. In K. Oberauer, *Working Memory* (pp. 116–149). Oxford University Press. https://doi.org/10.1093/oso/9780198842286.003.0005

Raghubar, K. P., Barnes, M. A., & Hecht, S. A. (2010). Working memory and mathematics: A review of developmental, individual difference, and cognitive approaches. *Learning and Individual Differences*, *20*(2), 110–122. https://doi.org/10.1016/j.lindif.2009.10.005

Semenza, C., Miceli, L., & Girelli, L. (1997). A Deficit for Arithmetical Procedures: Lack of Knowledge or Lack of Monitoring? *Cortex*, *33*(3), 483–498. https://doi.org/10.1016/S0010-9452(08)70231-4

Siegmund, J. (2016). Program Comprehension: Past, Present, and Future. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 13–20. https://doi.org/10.1109/SANER.2016.35

Stanovich, K. E. (2009). Matthew Effects in Reading: Some Consequences of Individual Differences in the Acquisition of Literacy. *Journal of Education*, *189*(1–2), 23–55. https://doi.org/10.1177/002205740918900l-204

Vergauwe, E., Souza, A. S., Langerock, N., & Oberauer, K. (2023). *The effect of instructed refreshing on working memory: Is the memory boost a function of refreshing frequency or refreshing duration?* [Preprint]. PsyArXiv. https://doi.org/10.31234/osf.io/6f4jp

Verhaeghen, P., Cerella, J., & Basak, C. (2004). A Working Memory Workout: How to Expand the Focus of Serial Attention From One to Four Items in 10 Hours or Less. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *30*(6), 1322–1337. https://doi.org/10.1037/0278-7393.30.6.1322

Yuan, K., Steedle, J., Shavelson, R., Alonzo, A., & Oppezzo, M. (2006). Working memory, fluid intelligence, and science learning. *Educational Research Review*, *1*(2), 83–98. https://doi.org/10.1016/j.edurev.2006.08.005

Zviran-Ginat, S. (2022). *How are Calculation Errors Related to Working Memory?* Unpublished MA dissertation, Tel Aviv University.

בית הספר סגול למדעי המוח

המחלקה למדעי המוח הקוגניטיביים

איך זיכרון פעיל תומך בתכנות, והאם תכנות תומך בו בחזרה?

מאת

ליהי כץ

208481119

החיבור בוצע בהנחיית ד״ר דרור דותן

ספטמבר 2023