

# סדנת Code Review



## רקע

הכול התחיל שבוע שעבר. השמש זרחה בחוץ, הנרקיסים בדיוק החלו את פריחתם לקראת בואו של האביב, ואני ישבתי לי בבית. פתאום הפלאפון שלי צלצל.

"הלו? אפשר לדבר עם המנהל של תוכנית מגשימים?"

"כן, מדבר"

"שלום! כאן סא"ל ד', מפקדת מערך פיתוח התוכנה של חיל הים," - ההתרגשות הייתה ניכרת בקולה - "בשנה האחרונה השקענו מלא משאבים בפיתוח תוכנה **סופר מסווגת** לספינות הטיילים שלנו".

"**סופר מסווגת?! שאלתי.**"

"כן, **סופר דופר מסווגת עם פצפוצים מלמעלה**. בקיצור, פיתחנו מערכת שתעזור לספינות שלנו לנווט את דרכן למדינות האויב. זו מערכת ממש משוכללת שמקבלת את: כיוון הרוח, כיוון הגלים, והכיוון שאליו החרטום של הספינה מכוון – ומחשבת את הדרך הקצרה ביותר אל האויב".

"וואו זה נשמע ממש מגניב! אבל למה את מספרת את זה לי?" – שאלתי בפליאה.

"אנחנו צריכים את העזרה של תוכנית מגשימים! צוות הפיתוח שלנו בדיוק סיים את הקוד של המערכת, אבל לצערנו הם לא עברו סדנת **Code Review** בקורס שלהם, והקוד שלהם לא... מי יודע מה. אפילו... ממוצע מינוס. אמרתי לצוות עוד מהתחלה שיכתבו 'קוד נקי', אבל הם לא הבינו למה התכוונתי, וכתבו קוד דיי מלוכלך".

"אוי זה נשמע נורא! אבל איך אנחנו יכולים לעזור?"

"הקוד כבר מוכן וגם עובד, אבל אנחנו מפחדים לשלוח את המערכת הזו לכל ספינות הטיילים של חיל הים מבלי שהוא נבדק בצורה טובה. ובדיוק בשביל זה אנחנו צריכים את התוכנית שלכם! בתוכנית מגשימים יש מלא מתכנתים ומתכנתות שלמדו לכתוב קוד נקי ואפילו לעשות Code Review איכותי. אז נשמח אם תוכלו לבדוק את הקוד שלנו!"

"כמובן! תוכנית מגשימים לשירותך, סא"ל ד'! נעשה הכול כדי לעזור למדינת ישראל!"

## מה עושים בתרגיל?

מנכ"ל התוכנית שלח לנו את הקוד של המערכת כדי שנעשה לו Code Review, אבל רגע לפני שאנחנו צוללים למים (הבנתם? כי חיל הים... 🤪) ועוברים על פרויקט תוכנה גדול – אנחנו רוצים לתרגל קצת את מה שלמדנו בשיעור, ורק אז נוכל לעבור על הקוד של המערכת.

לכן הסדנה מתחלקת ל-3 חלקים:

- 1) עושים CR לקוד שמפר את העקרונות של קוד נקי שלמדנו בשיעור ⚠️.
- 2) עושים CR לקוד שמכיל כמה טעויות נפוצות בקוד 🚫.
- 3) עושים CR לקוד של חיל הים 🚢 – מערכת ניווט ספינות.

## איך מתחילים?

יצירת סביבת העבודה

לפני שנתחיל, וודאו שאתם יודעים איך עושים Import לפרויקט באתר GitLab. תוכלו לצפות בסרטון שנמצא בתיקיית השיעור ב-Drive – שמדגים איך עושים CR בממשק של GitLab.

בתרגיל אנחנו נעשה CR לקוד שנמצא בתוך Merge Request בתוך פרויקט באתר GitLab.

**משימה** - עשו Import לקובץ **CodeReviewExercise.tar.gz** (שנמצא בתיקיית השיעור) – בעצם אתם יוצרים פרויקט חדש באתר GitLab שמבוסס על פרויקט אחר.

**משימה** - הוסיפו את היוזר של קורס עקרונות (נקרא Ekronot) לפרויקט שיצרתם (בדיוק כפי שעשיתם בכל התרגילים הקודמים).

**משימה** - היכנסו לדף Merge Requests של הפרויקט. זה מה שאנחנו רואים:

<b>01 Clean Code Principles</b> !7 · opened 2 hours ago by Ariel 🦋 develop
<b>02 General Mistakes</b> !6 · opened 2 hours ago by Ariel 🦋 develop
<b>03 Boat Navigation System</b> !4 · opened 3 hours ago by Ariel 🦋 develop

וודאו שיש 3 Merge Request-ים פתוחים (כמו בתמונה).

מעולה! סיימנו עם ההכנות.

## דגשים לתרגיל

1. **נסמן קטעי קוד** בתוך ההערה שלנו בין התווים `` (כפי שלמדנו בהדגמה) – כך קטעי הקוד ייראו בצורה שונה מטקסט רגיל, ויהיה נוח יותר לקרוא את ההערה.

2. **נכתב בהרחבה!** הדרך הכי טובה להבין נושא היא להסביר אותו למישהו אחר, ולכן אנחנו רוצים להסביר את העיקרון בהרחבה.
3. **נשתמש בגוגל** אם יש משהו שאנחנו לא כל כך הבנו בשיעור. אבל תכלס אפשר גם פשוט לחזור על השקופיות במצגת.
4. **נוסיף הערה על כל שורה שמפירה עיקרון כלשהו.** לא מפספסים שום שורה שדורשת תיקון! בכל קובץ יש מספר הפרות של העקרונות השונים, מצאו את כולם.
5. אחרי שנסיים להעיר על כל הקוד ב-Merge Request – **לא נמזג את ה-Merge Request** (כדי שהמדריך יוכל לראות את ההערות שלנו).

## מוכנים? בואו נתחיל!

### חלק 1 – עקרונות קוד נקי

בואו ניכנס ל-Merge Request הראשון – **01 Clean Code Principles**.

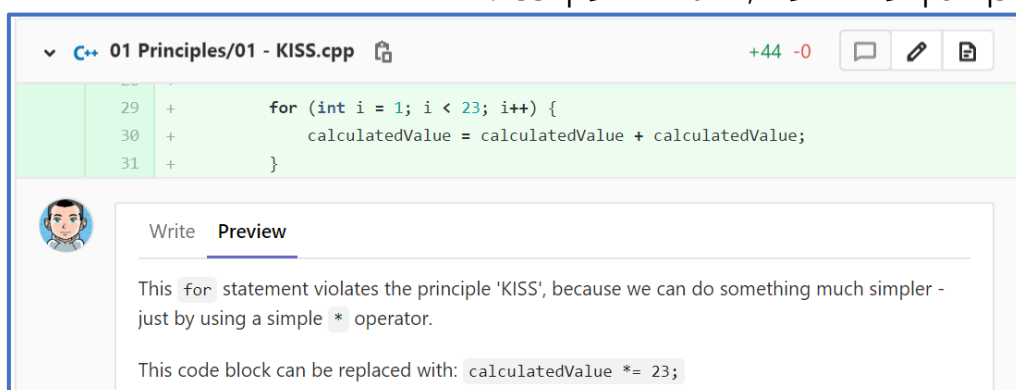
נעבור לטאב **Changes** –

Overview 0 Commits 1 Changes 6

ב-Merge Request הזה ישנם 6 קבצים. כל קובץ עוסק בעיקרון אחד מתוך עקרונות קוד נקי שלמדנו בשיעור. בכל קובץ ישנן **מספר הפרות** של אותו עיקרון – ואנחנו צריכים למצוא ולהעיר על כל ההפרות.

**המשימה שלנו** - לעבור על הקוד שבכל קובץ, ולמצוא את ההפרה של העיקרון. נוסיף Comment על **כל** המקומות שמפרים את העיקרון, ונסביר למה הקוד הזה מפר את העיקרון.

**לדוגמה**, בקובץ הראשון – **"01 - KISS.cpp"** ישנן מספר הפרות של העיקרון **KISS** (עיקרון הפשטות). לדוגמה, בשורה 29 (29-31) יש לולאת for שפשוט מבצעת הכפלה של המספר פי 23. לכן נוסיף על זה הערה, שתיראה בערך ככה:



The screenshot shows a code review interface for a file named "01 Principles/01 - KISS.cpp". The code snippet is as follows:

```
29 + for (int i = 1; i < 23; i++) {
30 +     calculatedValue = calculatedValue + calculatedValue;
31 + }
```

Below the code, there is a comment section with a "Write" tab and a "Preview" tab. The "Preview" tab is active, showing the following text:

This `for` statement violates the principle 'KISS', because we can do something much simpler - just by using a simple `*` operator.

This code block can be replaced with: `calculatedValue *= 23;`

## חלק 2 – טעויות נפוצות בקוד

נעבור ל-Merge Request השני – "02 General Mistakes".

ב-MR הזה ישנם 4 קבצים שמכילים טעויות נפוצות בקוד שלנו. לפני שניגש להעיר על הקוד, נלמד קודם על כמה טעויות כאלה:

### קוד חץ (Arrow Code)

קראו על הטעות הזו במאמר הבא:

<https://blog.codinghorror.com/flattening-arrow-code/>

### שיום (Naming)

כפי שלמדנו בשיעור, שיום היא פעולה בה ניתנים שמות לעצמים או למושגים אחרים. בהקשר שלנו הכוונה היא מתן שמות משמעותיים למחלקות, פונקציות, משתנים וכו'. כפי שאתם בוודאי כבר יודעים, שיום הוא אחד הדברים החשובים בעת כתיבת קוד נקי קריא וקל לתחזוקה.

### מספרי קסם

אלו מספרים שמשמעותם הייתה ידועה לכותב הקוד בזמן כתיבת הקוד, אך אדם שקורא את הקוד כעבור זמן – לא יבין מה המספר אומר, ומאיפה הוא לקוח. "מספר קסם" זה מספר שמופיע בקוד שלנו, ולא ברור למה דווקא המספר הזה נבחר, ולא מספר אחר.

הדרך הנכונה היא לרשום את המספר כ-define, const או enum – ובכך נוכל להשתמש במספר פעמים נוספות ברחבי הקוד, וגם נוכל לשנות אותו בקלות אם נצטרך.

### הערות (Comments)

ישנן מספר טעויות נפוצות ביצירה של הערות –

**הערות מיותרות** – הערות שמסבירות דברים מובנים מאליהם.

**הערות הסבר מיותרות** - הערות הסבר הן חשובות כאשר נרצה לתאר אלגוריתם מורכב, קוד ברמה נמוכה (Low Level) וכו'. אך לפעמים הן מסבירות דברים שאפשר להבין בקלות אם פשוט נקרא את הקוד.

כלל האצבע לפני כתיבת הערה – ננסה לקרוא את הקוד שלנו בקול. אם הבנו את מה שאמרנו – מצוין! סימן שהקוד מספיק מובן, ואין צורך להוסיף הערה. אם לא הבנו – סימן שצריך להוסיף הערה (קצרה ועניינית, כמובן).

**התנצלויות/אזהרות** - לפעמים נרגיש צורך להתנצל על קטע קוד שלא עובד טוב, או להזהיר את המתכנת הבא שלא ישנה (חס וחלילה) איזו שורת קוד. במקום זה... נתקן את הקוד!

**שפה לא נאותה/בדיחות** - נשאיר את הלא-פורמלי מחוץ לתחום. אם עולה הצורך להוסיף הערה לקוד, חשוב שנדאג שהיא תהיה קצרה ועניינית ככל הניתן. כבדו את עצמכם ואת המתכנתים האחרים שיקראו ויתחזקו את הקוד שלכם.

**קוד שנמצא בהערה** – לפעמים אנחנו מסמנים חלק מהקוד שלנו בהערה כדי לבדוק משהו. זה בסדר במהלך התהליך של הפיתוח, אך קוד בהערה לא אמור להגיע ל-Merge Request! מחקו קוד שנמצא בהערה.

כעת הגיע הזמן לבצע את המשימה של החלק הזה, בחלק הזה ניעזר ב-ChatGPT...

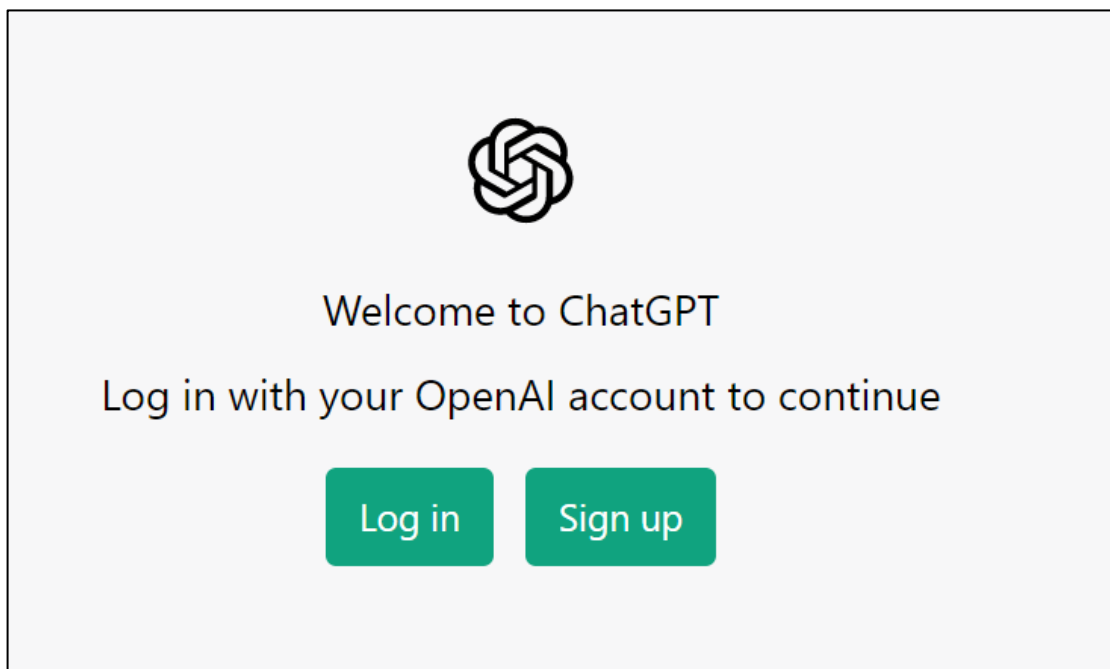


**משימה** - ראשית נעבור על הקוד שנמצא בקבצים של ה-Merge Request הנוכחי, ונחפש בכל קובץ את הטעויות שעליהן למדנו. נרשום לעצמנו טעויות שמצאנו.

אחרי שעברנו בעצמנו על הקוד, ניכנס לאתר של חברת OpenAI

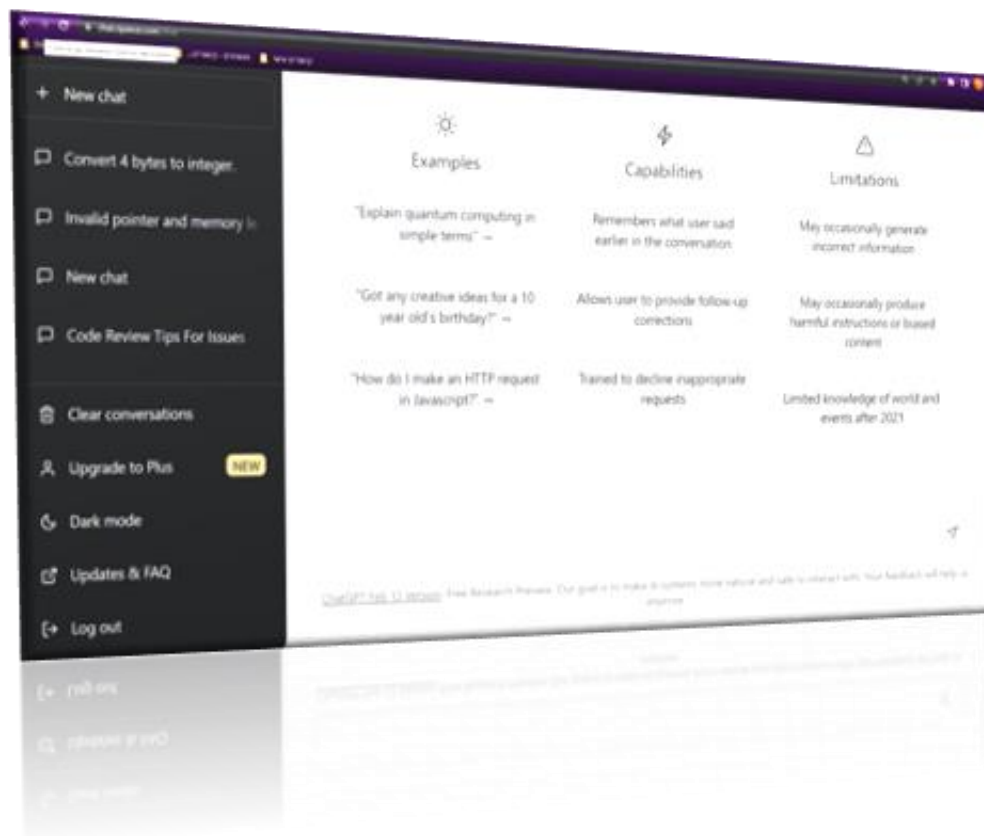
<https://chat.openai.com>

אם כבר השתמשתם ב-ChatGPT בעבר, השתמשו בחשבון שדרכו התחברתם/ במידה וזו הפעם הראשונה, ליחצו על Sign Up והירשמו



אחרי שהתחברנו, נוכל סוף סוף לדבר עם ChatGPT.

אז קצת על ChatGPT...



ChatGPT מאפשר לנו לקבל המון מידע באמצעות ניהול שיחה, ובשביל הרבה אנשים זו דרך מאוד נוחה להתנהל.

מעבר לדוגמאות קוד מציאת מתכונים, כתיבת מאמרים, והסברים על תופעות טבעיות; ה-ChatGPT גם יכול לעזור לנו להתכונן לראיון עבודה, לקבל הערות על מצגות ונאומים שאנחנו מתכוונים להציג, ואפילו לעזור לנו ללמוד שפות חדשות.

במידה ותרצו, הנה GitHub repo נחמד שמראה שימושים מאוד יעילים ב-ChatGPT, באתר אפשר למצוא פרומפטים (המשפטים שאנחנו מכניסים לצ'אט) בכל מיני נושאים, ודוגמאות מכל מיני תחומים בחיים.

<https://github.com/f/awesome-chatgpt-prompts>



ובחזרה לקורס עקרונות, ChatGPT יכול לתת לנו Code Review על הקוד שאנחנו כותבים.

לדוגמא, נסתכל על הקוד הבא:

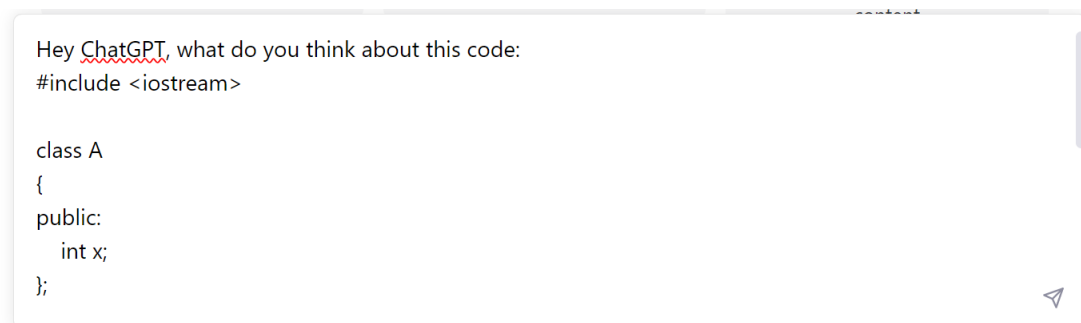
```
#include <iostream>

class A
{
public:
    int x;
};

A* foo()
{
    A a;
    return &a;
}

int main()
{
    int* a = new int;
    return 0;
}
```

נוכל להעתיק את הקוד ולבקש מ-ChatGPT שייתן לנו הערות.



[ChatGPT Feb 13 Version](#). Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

בדוגמא כאן, רשמנו את הפרומפט "Hey ChatGPT, what do you think about this code" ירדנו שורה באמצעות Alt+Enter ואז הדבקנו את הקוד שהעתקנו מ-Visual Studio.

במקרה הזה ChatGPT הסביר לנו על כמה בעיות שיש בתכנית כמו דליפת זיכרון, החזרה של כתובת של משתנה מקומי, ואפילו הציע לנו קוד עם פיתרון לבעיות.

משימה - הריצו את הדוגמא והגישו את התשובה ש-ChatGPT החזיר.

עוד משהו נחמד שכדאי להכיר, אפשר לבקש מ-ChatGPT להתייחס לקטגוריות בדיקה ספציפיות...

לדוגמה אפשר להכניס ל-ChatGPT את הפרומפט הבא:

Hey Chat GPT please review this code:

<OUR CODE>

Address the following topics:

Code errors

Design

Performance

Readability

Bad Practices

במקרה הזה הפלט שנקבל חזרה מ-ChatGPT יחולק לקטגוריות, ובכל אחת ChatGPT יכתוב לנו כמה הערות בנושא.

לדוגמה: אם תריצו את הדוגמה שלמעלה, תוכלו לראות ש-ChatGPT כתב כמה הערות בנושא "קריאות" (Readability)

Readability:

1. The variable names used in the code are not very descriptive, which can make it difficult to understand what the code is doing.
2. There is no documentation or comments in the code, which can make it hard to understand the purpose of the code.

ובקיצור, אפשר לראות כמה זה יכול לעזור לנו לקבל חוות דעת על הקוד גם מבינה מלאכותית... 🔍🤖

**משימה** – יש להשתמש ב-ChatGPT כדי לקבל Code Review על הקוד שנמצא בקבצים של ה-Merge Request הנוכחי.  
נסו למצוא כמה שיותר מהטעויות שזיהינו קודם בעצמנו.  
הגישו את התשובה ש-ChatGPT החזיר, וגם את הפרומפט שאותו רשמתם/ן.



## חלק 3 – מערכת ניווט לספינות

אחרי שעברנו על העקרונות הבסיסיים ותרגלנו את העקרונות שלמדנו בשיעור, הגיע הזמן לעבור למשימה האמיתית שלנו – Code Review לקוד של הפרויקט של חיל הים.

ניכנס ל-Merge Request השלישי – "03 Boat Navigation System".

נעבור על כל הקוד, ונוסיף הערות במקומות שבהם אנחנו מוצאים הפרה של כלל מסוים.

**שימו לב:** הקוד לא מפר את כל הכללים שלמדנו (בכל זאת, צוותי הפיתוח של הצבא לא כאלה גרועים...), ולכן אל תשאפו למצוא הפרה של כל העקרונות שלמדנו בשיעור... אבל בהחלט תשאפו למצוא כמה שיותר!

**כדי להקל עליכם את העבודה, הכנו Cheat Sheet של כל הכללים והעקרונות שעליהם למדנו: (הכללים שאנחנו צריכים לחפש בקוד)**

- עקרון הפשטות
- עקרון אי-חזרה
- עקרון האחרייות היחידה
- עקרון הפתוח-סגור
- עקרון ההחלפה של ליסקוב
- עקרון הפרדת הממשקים
- קונבנציות בשפת ++c שלמדנו בקורס
- שגיאות לוגיות
- מקרי קצה שלא נבדקו
- Naming
  - שמות קצרים מדי
  - שמות צריכים להיות ניתנים להגייה
  - שמות שדורשים תיעוד
- תיעוד (הערות)
  - קצר וברור
  - לא מתעדים דברים מובנים מאליהם
  - קוד שנמצא בהערה
  - יש תיעוד בקטעי קוד לא מובנים

## הגשה של הסדנה

את תרגול הכיתה צריך להגיש למדריך לבדיקה.

וודאו שהיוזר של קורס עקרונות (Ekronot) נמצא ב-Members של הפרויקט שלכם/ן.

עליכם להגיש קישור ל-Repo של הפרויקט.

בתוך ה-Repo אמורים להיות **3 Merge Requests פתוחים** – כפי שהיו בהתחלה. וודאו שלא מיזגתם/ן אף אחד מהם ל-Develop, ובתוך כל MR מופיעות ההערות שהוספתם.

**כל הכבוד! סא"לד' וצוות הפיתוח שלה מודים לכם מאוד!**

ועכשיו אפשר לעבור לתרגיל הבית...

**בהצלחה!**

