

# מדריך הערכה עצמית מקיף

## Comprehensive Self-Assessment Guide

Dr. Yoram Segal

Version 2.0  
22-11-2025

### תוכן העניינים

3	1	מבוא כללי
3	1.1	מטרת המדריך
3	2.1	כיצד להשתמש במדריך זה
4	I	עקרונות ההערכה העצמית האקדמית
4	2	עקרונות יסוד
4	1.2	עקרון מרכזי: רמת הדקדקנות בבדיקה תושפע מהציון העצמי
4	3	המלצה לשלבים לביצוע הערכה עצמית
4	1.3	שלב 1: הבנת הקריטריונים והתקנים
4	2.3	שלב 2: מיפוי העבודה שלכם מול הקריטריונים
4	1.2.3	תיעוד פרויקט (Project Documentation) -- 20%
5	2.2.3	README ותיעוד קוד (Code Documentation) -- 15%
5	3.2.3	מבנה פרויקט ואיכות קוד (Project Structure & Code Quality) -- 15%
6	4.2.3	קונפיגורציה ואבטחה (Configuration & Security) -- 10%
6	5.2.3	בדיקות ואיכות (Testing & QA) -- 15%
7	6.2.3	מחקר וניתוח (Research & Analysis) -- 15%
7	7.2.3	ממשק משתמש והרחבה (UI/UX & Extensibility) -- 10%
7	3.3	שלב 3: ניתוח עומק וייחודיות
9	4	קביעת הציון העצמי -- מדריך לפי רמות
9	1.4	רמה 1: ציון 60--69 (מעבר בסיסי)
9	2.4	רמה 2: ציון 70--79 (טוב)
9	3.4	רמה 3: ציון 80--89 (טוב מאוד)
01	4.4	רמה 4: ציון 90--100 (מצוינות יוצאת דופן)
11	5	טבלת הערכה עצמית מסכמת
21	6	טופס הגשת הערכה עצמית
21	1.6	הצדקה להערכה העצמית (חובה -- 200--500 מילים)
21	2.6	רמת הדקדקנות המבוקשת בבדיקה
31	3.6	הצהרת יושר אקדמי (Academic Integrity Declaration)

41	7	טיפים להערכה עצמית מוצלחת
41	1.7	עשו (DO):
41	2.7	אל תעשו (DON'T):
51	8	שאלות נפוצות (FAQ)
61	II	בדיקה טכנית מפורטת של הקוד
61	9	מבוא לבדיקה הטכנית
61	01	בדיקה א: ארגון הפרויקט כחבילה
61	1.01	רקע
61	2.01	רשימת בדיקה: מבנה החבילה
71	3.01	דוגמאות למבנה נכון
71	4.01	הערות נוספות
81	11	בדיקה ב: שימוש במעבדים מרובים וחטי ביצוע מרובים
81	1.11	רקע
81	2.11	רשימת בדיקה: Multiprocessing
81	3.11	רשימת בדיקה: Multithreading
91	4.11	שיקולים נוספים
91	5.11	הערות נוספות
02	21	בדיקה ג: עיצוב מבוסס אבני בניין
02	1.21	רקע
02	2.21	עקרונות עיצוב
02	3.21	רשימת בדיקה: זיהוי אבני בניין
02	4.21	רשימת בדיקה: נתוני קלט (Input Data)
12	5.21	רשימת בדיקה: נתוני פלט (Output Data)
12	6.21	רשימת בדיקה: נתוני הגדרה (Setup Data)
22	7.21	דוגמה לאבן בניה טובה
22	8.21	הערות נוספות
32	III	סיכום והמלצות
32	31	ציון כללי
32	1.31	ציון אקדמי (מהחלק הראשון)
32	2.31	ציון טכני (מהחלק השני)
32	3.31	ציון כולל
32	41	תחומים לשיפור
42	51	תכנית פעולה
42	61	לסיכום

## **1 מבוא כללי**

מדריך זה מיועד לסייע לכם בביצוע הערכה עצמית מקיפה של פרויקט הקוד שלכם, הן מהבחינה האקדמית והן מהבחינה הטכנית. ההערכה העצמית היא חלק חיוני מתהליך הלמידה והפיתוח, ומאפשרת לכם לזהות נקודות חוזקה וחולשה, לפתח חשיבה רפלקטיבית, ולשפר את איכות העבודה.

### **1.1 מטרת המדריך**

מדריך זה משלב שני ממדים:

1. **הערכה אקדמית:** מסגרת להערכה עצמית של איכות העבודה הכללית, התיעוד, והמחקר

2. **הערכה טכנית:** בדיקה מפורטת של היבטים טכניים כגון ארגון קוד, multipro-cessing, ועיצוב מודולרי

### **2.1 כיצד להשתמש במדריך זה**

עברו על כל חלק בתשומת לב, וענו על השאלות בהתאם לפרויקט שלכם. סמנו את הפריטים שהושלמו, וציינו את אלו הדורשים שיפור. הקדישו זמן לרפלקציה ולחשיבה ביקורתית על העבודה שלכם.

## חלק I

# עקרונות ההערכה העצמית האקדמית

## 2 עקרונות יסוד

הערכה עצמית היא תהליך אקדמי חשוב המעודד חשיבה רפלקטיבית, אחריות אישית ומודעות למהלך הלמידה. במסגרת קורס זה, אתם מוזמנים לקבוע את הציון שאתם סבורים שמגיע לכם. ציון זה ייקבע לפי הערכתכם העצמית את איכות העבודה שהגשתם ביחס לקריטריונים שנקבעו.

### 1.2 עקרון מרכזי: רמת הדקדקנות בבדיקה תושפע מהציון העצמי

ככל שהציון העצמי גבוה יותר, כך רמת הדקדקנות והקפדנות בבדיקה תהיה גבוהה יותר. זהו עקרון של "הבטחה--ציפייה" (Contract-Based Grading):

□ **ציון עצמי גבוה (100--90):** בדיקה מאוד דקדקנית ומעמיקה, חיפוש "פילים בקנה", בדיקת כל פרט קטן

□ **ציון עצמי בינוני (89--75):** בדיקה סבירה ומאוזנת עם קריטריונים ברורים

□ **ציון עצמי נמוך יותר (74--60):** בדיקה גמישה, אוהדת ומכילה -- אם יש הגיון בהגשה והיא סבירה

## 3 המלצה לשלבים לביצוע הערכה עצמית

אין חובה למלא ולהגיש זאת ביחד עם הגשת המטלה, אך מומלץ מאוד.

### 1.3 שלב 1: הבנת הקריטריונים והתקנים

לפני שתוכלו להעריך את עבודתכם, עליכם להבין במדויק את הקריטריונים שעל פיהם העבודה תוערך:

□ קראו בעיון את מסמך ההנחיות להגשת תוכנה

□ זהו את כל המרכיבים הנדרשים (תיעוד, קוד, בדיקות, ניתוח וכו')

□ הבינו את רמת האיכות המצופה בכל קריטריון

□ שימו לב להבחנות בין רמות איכות שונות

### 2.3 שלב 2: מיפוי העבודה שלכם מול הקריטריונים

השתמשו ב-checklist הבא כדי לבדוק אילו מרכיבים כללתם בעבודה:

#### 1.2.3 תיעוד פרויקט (Project Documentation) -- 20%

#### PRD (Product Requirements Document)

□ □ תיאור ברור של מטרת הפרויקט ובעיית המשתמש

□ □ יעדים מדידים ומדדי הצלחה (KPIs)

□ □ דרישות פונקציונליות ולא-פונקציונליות מפורטות

☐ תלויות, הנחות ומגבלות

☐ ציר זמן ואבני דרך

#### תיעוד ארכיטקטורה

☐ תרשימי בלוקים (C4 Model, UML)

☐ ארכיטקטורה תפעולית

☐ החלטות ארכיטקטוניות (ADRs)

☐ תיעוד API וממשקים

ציון עצמי בקטגוריה זו: \_\_\_\_\_/20

### 2.2.3 README ותיעוד קוד (Code Documentation) -- 15%

#### README מקיף

☐ הוראות התקנה שלב-אחר-שלב

☐ הוראות הפעלה מפורטות

☐ דוגמאות הרצה וצילומי מסך

☐ מדריך קונפיגורציה

☐ troubleshooting

#### איכות הערות בקוד

☐ Docstrings לכל פונקציה/class/מודול

☐ הסברים על החלטות עיצוב מורכבות

☐ שמות משתנים ופונקציות תיאוריים

ציון עצמי בקטגוריה זו: \_\_\_\_\_/15

### 3.2.3 מבנה פרויקט ואיכות קוד (Project Structure & Code Quality) -- 15%

#### ארגון פרויקט

☐ מבנה תיקיות מודולרי וברור (src/, tests/, docs/, data/, results/, config/, as-sets/)

☐ הפרדה בין קוד, נתונים ותוצאות

☐ קבצים לא עולים על 150 ~ שורות

☐ naming conventions עקביות

## איכות קוד

☐ פונקציות קצרות וממוקדות (Single Responsibility)

☐ הימנעות מקוד כפול (DRY)

☐ עקביות בסגנון קוד

ציון עצמי בקטגוריה זו: \_\_\_\_\_ /15

## 4.2.3 קונפיגורציה ואבטחה (Configuration & Security) -- 10%

### ניהול קונפיגורציה

☐ קבצי קונפיגורציה נפרדים (.env, .yaml, .json)

☐ אין קבועים hardcoded בקוד

☐ קבצי דוגמה (.env.example)

☐ תיעוד פרמטרים

### אבטחת מידע

☐ אין API keys בקוד המקור

☐ שימוש במשתני סביבה

☐ .gitignore מעודכן

ציון עצמי בקטגוריה זו: \_\_\_\_\_ /10

## 5.2.3 בדיקות ואיכות (Testing & QA) -- 15%

### כיסוי בדיקות

☐ unit tests עם כיסוי +70% לקוד חדש

☐ בדיקות edge cases

☐ דוחות כיסוי (coverage reports)

### טיפול בשגיאות

☐ תיעוד edge cases עם תיאור ותגובה

☐ error handling מקיף

☐ הודעות שגיאה ברורות

☐ לוגים לצורך debugging

### תוצאות בדיקה

☐ תיעוד תוצאות צפויות

☐ דוחות automated testing

ציון עצמי בקטגוריה זו: \_\_\_\_\_ /15

### 6.2.3 מחקר וניתוח (Research & Analysis) -- 15%

#### ניסויים ופרמטרים

- ☐ ☐ ניסויים שיטתיים עם שינוי פרמטרים
- ☐ ☐ ניתוח רגישות (sensitivity analysis)
- ☐ ☐ טבלת ניסויים עם תוצאות
- ☐ ☐ זיהוי פרמטרים קריטיים

#### מחברת ניתוח

- ☐ ☐ Jupyter Notebook או כלי דומה
- ☐ ☐ ניתוח מתודי ומעמיק
- ☐ ☐ נוסחאות מתמטיות ב- $\text{LaTeX}$  (אם רלוונטי)
- ☐ ☐ אסמכתאות לספרות אקדמית

#### הצגה ויזואלית

- ☐ ☐ גרפים איכותיים (bar charts, line charts, heatmaps, וכו')
- ☐ ☐ תוויות ומקרא ברורים
- ☐ ☐ רזולוציה גבוהה
- ציון עצמי בקטגוריה זו: \_\_\_\_\_ /15

### 7.2.3 ממשק משתמש והרחבה (UI/UX & Extensibility) -- 10%

#### ממשק משתמש

- ☐ ☐ ממשק ברור ואינטואיטיבי
- ☐ ☐ צילומי מסך ותיעוד workflow
- ☐ ☐ נגישות (accessibility)

#### הרחבה (Extensibility)

- ☐ ☐ נקודות הרחבה (extension points/hooks)
- ☐ ☐ תיעוד פיתוח plugins
- ☐ ☐ ממשקים ברורים
- ציון עצמי בקטגוריה זו: \_\_\_\_\_ /10

### 3.3 שלב 3: ניתוח עומק וייחודיות

ענו על השאלות הבאות להערכת העומק והייחודיות של העבודה:

### עומק טכני

- ☐ ☐ השתמשתי בטכניקות מתקדמות של סוכני AI?
- ☐ ☐ הוספתי ניתוח מתמטי או תיאורטי?
- ☐ ☐ ביצעתי מחקר השוואתי בין גישות שונות?

### ייחודיות וחדשנות

- ☐ ☐ הפרויקט כולל רעיונות מקוריים או גישה חדשנית?
- ☐ ☐ פיתחתי פתרון לבעיה מורכבת או מאתגרת?
- ☐ ☐ הוספתי ערך מעבר לדרישות הבסיסיות?

### ספר הפרומפטים

- ☐ ☐ תיעדתי את תהליך הפיתוח עם AI?
- ☐ ☐ כללתי דוגמאות לפרומפטים משמעותיים?
- ☐ ☐ הוספתי best practices מהניסיון?

### עלויות ותמחור

- ☐ ☐ חישבתי שימוש ב-tokens?
- ☐ ☐ הצגתי טבלת עלויות מפורטת?
- ☐ ☐ הצעתי אסטרטגיות אופטימיזציה?



## 4 קביעת הציון העצמי -- מדריך לפי רמות

### 1.4 רמה 1: ציון 69--60 (מעבר בסיסי)

**תיאור:** הגשה סבירה שמכסה את הדרישות המינימליות מאפיינים:

- קוד עובד ומבצע את המשימות הנדרשות
  - תיעוד בסיסי (README עם הוראות התקנה והפעלה)
  - מבנה פרויקט הגיוני אך לא בהכרח מושלם
  - בדיקות בסיסיות או כיסוי חלקי
  - תוצאות קיימות אך ללא ניתוח מעמיק
- רמת הבדיקה שתקבלו:** גמישה, אוהדת ומכילה. הבודקים יחפשו את ההגיון והסבירות בעבודה ולא יתעכבו על פרטים קטנים.
- המלצה:** בחרו ברמה זו אם השקעתם מאמץ סביר אך יודעים שהעבודה לא מושלמת, או אם הזמן היה מוגבל.

### 2.4 רמה 2: ציון 79--70 (טוב)

**תיאור:** עבודה איכותית עם תיעוד טוב ומבנה מסודר מאפיינים:

- קוד מסודר עם הערות והפרדה למודולים
  - תיעוד מקיף: README טוב, תיעוד ארכיטקטורה, PRD בסיסי
  - מבנה פרויקט נכון עם הפרדה בין קוד, נתונים ותוצאות
  - בדיקות עם כיסוי 50--70%
  - ניתוח תוצאות עם גרפים בסיסיים
  - קונפיגורציה נכונה ואבטחת API keys
- רמת הבדיקה שתקבלו:** סבירה ומאוזנת. הבודקים יבדקו עמידה בקריטריונים עיקריים אך יתנו מרחב לשגיאות קטנות.
- המלצה:** בחרו ברמה זו אם עמדתם ברוב הדרישות ויצרתם עבודה מסודרת ואיכותית.

### 3.4 רמה 3: ציון 89--80 (טוב מאוד)

**תיאור:** עבודה מצוינת ברמה אקדמית גבוהה מאפיינים:

- קוד מקצועי עם modularity גבוהה והפרדת אחריות
- תיעוד מלא ומפורט: PRD מקיף, ארכיטקטורה עם תרשימי C4, README ברמת user manual
- מבנה פרויקט מושלם לפי best practices

- בדיקות מקיפות עם כיסוי 70--85%
  - מחקר ממשי: ניתוח רגישות פרמטרים, מחברת ניתוח עם נוסחאות
  - הצגה ויזואלית מרשימה של תוצאות
  - ממשק משתמש איכותי
  - עלויות מתועדות וניתוח אופטימיזציה
- רמת הבדיקה שתקבלו:** מעמיקה ומדוקדקת. הבודקים יבדקו התאמה מלאה לקריטריונים ויקפידו על רמת איכות גבוהה.
- המלצה:** בחרו ברמה זו אם השקעתם מאמץ משמעותי, כיסיתם את כל הדרישות, וביצעתם מחקר אמיתי.

#### 4.4 רמה 4: ציון 90--100 (מצוינות יוצאת דופן)

**תיאור:** רמת MIT -- עבודה ברמת פרסום אקדמי או תעשייתי  
**מאפיינים:**

- קוד ברמת production עם extensibility, hooks, וארכיטקטורת plugins
  - תיעוד מושלם ומפורט בכל היבט: PRD מקיף, תיעוד ארכיטקטורה מלא, README מקצועי
  - עמידה מלאה בתקן ISO/IEC 25010
  - בדיקות מקיפות עם כיסוי +85%, edge cases מתועדים ומטופלים
  - מחקר מעמיק: ניתוח רגישות שיטתי, הוכחות מתמטיות, השוואה מבוססת-נתונים
  - ויזואליזציה ברמה גבוהה ביותר עם dashboard אינטראקטיבי
  - ספר פרומפטים מפורט ומתועד
  - ניתוח עלויות מקיף עם המלצות אופטימיזציה
  - חדשנות וייחודיות: רעיונות מקוריים, פתרון לבעיה מורכבת
  - תרומה לקהילה: קוד פתוח, תיעוד לשימוש חוזר
- רמת הבדיקה שתקבלו:** דקדקנית ביותר -- ``חיפוש פילים בקנה". הבודקים יבדקו כל פרט קטן, יחפשו חוסרים קטנטנים, ויקפידו על כל ``קוצו של יוד".
- אזהרה:** רמה זו מיועדת למי שבטוח לחלוטין שהעבודה ברמת מצוינות מקסימלית. אם יימצאו חוסרים, הציון עלול לרדת משמעותית.
- המלצה:** בחרו ברמה זו רק אם:
- כיסיתם את כל הקריטריונים ללא יוצא מן הכלל
  - ביצעתם בדיקה עצמית מעמיקה והכל מושלם
  - יש חדשנות וייחודיות משמעותית
  - אתם מוכנים לבדיקה מאוד קפדנית

## 5 טבלת הערכה עצמית מסכמת

השתמשו בטבלה הבאה כדי לחשב את הציון העצמי שלכם:

קטגוריה	משקל	הציון שלי	ציון משוקלל
תיעוד פרויקט (PRD, ארכיטקטורה)	20%	_____	_____
README ותיעוד קוד	15%	_____	_____
מבנה פרויקט ואיכות קוד	15%	_____	_____
קונפיגורציה ואבטחה	10%	_____	_____
בדיקות ואיכות (Testing & QA)	15%	_____	_____
מחקר וניתוח תוצאות	15%	_____	_____
ממשק משתמש והרחבה	10%	_____	_____
<b>סה"כ</b>	<b>100%</b>		

## 6 טופס הגשת הערכה עצמית

אנא מלאו את המידע הבא והגישו יחד עם הפרויקט:

שם הסטודנט/ים: \_\_\_\_\_  
שם הפרויקט: \_\_\_\_\_  
תאריך הגשה: \_\_\_\_\_  
הציון העצמי שלי: \_\_\_\_\_ /100

### 1.6 הצדקה להערכה העצמית (חובה -- 500--200 מילים)

בסעיף זה, הסבירו למה בחרתם בציון זה. כללו:

- ☐ **נקודות חוזק:** מה עשיתם במיוחד טוב? אילו מרכיבים הם ברמה גבוהה?
- ☐ **נקודות חולשה:** מה חסר או יכול היה להיות טוב יותר? (כנות מוערכת!)
- ☐ **השקעה:** כמה זמן ומאמץ השקעתם?
- ☐ **חדשנות:** האם יש משהו ייחודי או מיוחד בעבודה?
- ☐ **למידה:** מה למדתם מהפרויקט?

### 2.6 רמת הדקדקנות המבוקשת בבדיקה

על פי הציון העצמי שנתתי, אני מבין/ה שרמת הבדיקה תהיה:

- ☐ ☐ **60--69:** גמישה, אוהדת ומכילה -- בדיקת הגיון והתאמה בסיסית
- ☐ ☐ **70--79:** סבירה ומאוזנת -- בדיקת קריטריונים עיקריים
- ☐ ☐ **80--89:** מעמיקה ומדוקדקת -- בדיקה מלאה של כל הקריטריונים
- ☐ ☐ **90--100:** דקדקנית ביותר -- חיפוש "פילים בקנה", הקפדה על כל פרט

### 3.6 הצהרת יושר אקדמי (Academic Integrity Declaration)

אני מצהיר/ה בזאת ש:

- ☐ ☐ ההערכה העצמית שלי היא כנה ואמיתית
- ☐ ☐ בדקתי את העבודה מול כל הקריטריונים לפני קביעת הציון
- ☐ ☐ אני מודע/ת שציון עצמי גבוה יוביל לבדיקה דקדקנית יותר
- ☐ ☐ אני מקבל/ת את העובדה שהציון הסופי עשוי להיות שונה מהציון העצמי
- ☐ ☐ העבודה היא פרי עבודתי/נו (של הקבוצה) ואני/ו אחראי/ם לכל תוכנה

חתימה: \_\_\_\_\_ תאריך: \_\_\_\_\_

## **7 טיפים להערכה עצמית מוצלחת**

### **1.7 עשו (DO):**

- **היו כנים** -- הערכה עצמית מדויקת מועילה לכם יותר מציון מנופח
- **השתמשו בקריטריונים** -- עברו שיטתית על כל סעיף בהנחיות
- **תעדו את התהליך** -- שמרו רשימה של מה עשיתם ומה חסר
- **קבלו פידבק** -- שאלו חברים לעבור על העבודה לפני הגשה
- **הקדישו זמן לרפלקציה** -- חשבו מה למדתם והיכן אתם יכולים להשתפר

### **2.7 אל תעשו (DON'T):**

- **אל תנפחו ציון** -- ציון גבוה מדי יוביל לבדיקה קשה ואכזבה
- **אל תזלזלו בעבודה** -- גם אם לא מושלמת, ייתכן שהיא טובה יותר ממה שחושבים
- **אל תשכחו הצדקה** -- הסבר חסר יקשה על הבודקים להבין את ההערכה
- **אל תחכו לרגע האחרון** -- הערכה עצמית איכותית דורשת זמן
- **אל תשכחו את החתימה** -- הצהרת יושר אקדמי היא חובה

## 8 שאלות נפוצות (FAQ)

**ש: מה קורה אם הציון שאתן לעצמי יהיה שונה מהציון שהבודק/ת יתן/תן?**

**ת:** זה נורמלי לחלוטין. הערכה עצמית היא כלי למידה, לא קביעת ציון סופית. הבודק/ת ישקלו את ההערכה העצמית אך יקבעו את הציון הסופי על פי שיקול דעתם המקצועי.

**ש: האם כדאי תמיד לבחור בציון נמוך כדי לקבל בדיקה גמישה?**

**ת:** לא. ציון עצמי נמוך יכול להוביל לציון סופי נמוך יותר גם אם העבודה טובה. הבחירה צריכה לשקף את איכות העבודה בפועל, לא אסטרטגיה.

**ש: האם יש ערעור על הציון?**

**ת:** כן, אך הבסיס לערעור חייב להיות משמעותי. אם הערכתם את עצמכם בכנות, ערעורים יהיו נדירים.

**ש: האם אפשר לעדכן את הציון העצמי לאחר ההגשה?**

**ת:** לא. הציון העצמי הוא חלק מההגשה וקובע את רמת הבדיקה. שינוי לאחר מכן אינו אפשרי.

**ש: מה אם קשה לי להעריך את עצמי?**

**ת:** השתמשו ב-checklist, שאלו חברי קבוצה או עמיתים, והיעזרו בקריטריונים המפורטים. הערכה עצמית היא מיומנות שמשתפרת עם התרגול.

## חלק II

# בדיקה טכנית מפורטת של הקוד

## 9 מבוא לבדיקה הטכנית

חלק זה של המדריך מתמקד בהיבטים הטכניים של פרויקט הקוד, ומספק רשימות בדיקה מפורטות לארגון הפרויקט כחבילה, שימוש במעבדים מרובים וחוטי ביצוע, ועיצוב מבוסס אבני בניין.

## 01 בדיקה א: ארגון הפרויקט כחבילה

### 1.01 רקע

ארגון הקוד כחבילה (package) הוא עיקרון יסודי בפיתוח תוכנה מקצועי. חבילה מאורגנת נכון מאפשרת:

- שימוש חוזר בקוד במספר פרויקטים
- ניהול תלויות (dependencies) בצורה ברורה
- הפצה והתקנה פשוטה
- בדיקות (testing) מובנות

### 2.01 רשימת בדיקה: מבנה החבילה

עברו על הפריטים הבאים ובדקו את הפרויקט שלכם:

#### 1. קובץ `setup.py` או `pyproject.toml`:

- □ האם קיים קובץ הגדרת חבילה (`setup.py` או `pyproject.toml`)?
- □ האם הקובץ מכיל את כל המידע הנדרש (שם, גרסה, תלויות)?
- □ האם התלויות מפורטות בצורה מלאה עם מספרי גרסאות?

#### 2. קובץ `__init__.py`:

- □ האם קיים קובץ `__init__.py` בתיקייה הראשית של החבילה?
- □ האם הקובץ מייצא (export) את הממשקים הציבוריים של החבילה?
- □ האם קבועים כגון `__version__` מוגדרים בקובץ זה?

#### 3. מבנה תיקיות מאורגן:

- □ האם קוד המקור נמצא בתיקייה ייעודית (למשל `src/` או בשם החבילה)?
- □ האם הבדיקות (tests) נמצאות בתיקייה נפרדת?
- □ האם התיעוד (docs) נמצא בתיקייה נפרדת?

#### 4. נתיבים יחסיים:



- ☐ ☐ האם כל הייבואים (imports) בקוד משתמשים בנתיבים יחסיים או בשמות חבילות?
- ☐ ☐ האם הקוד נמנע משימוש בנתיבים מוחלטים (absolute paths)?
- ☐ ☐ האם קריאה וכתובה של קבצים נעשית ביחס לנתיב החבילה ולא למיקום קובץ ההרצה?

#### 5. מציין מיקום לקוד גיבוב (hash code placeholder):

- ☐ ☐ האם קיימים מיקומים בקוד המיועדים לחישוב גיבוב (hash)?
- ☐ ☐ האם המיקומים מסומנים בצורה ברורה (הערות או פונקציות ייעודיות)?
- ☐ ☐ האם יישום הגיבוב עקבי לאורך כל הפרויקט?

### 3.01 דוגמאות למבנה נכון

מבנה תיקיות מומלץ:

```
my_project/
  src/
    my_package/
      __init__.py
      core.py
      utils.py
  tests/
    __init__.py
    test_core.py
  docs/
  setup.py
  README.md
  requirements.txt
```

### 4.01 הערות נוספות

רשמו כאן הערות על ממצאים, בעיות שזוהו, או שיפורים נדרשים:

## 11 בדיקה ב: שימוש במעבדים מרובים וחטי ביצוע מרובים

### 1.11 רקע

שימוש במעבדים מרובים (multiprocessing) ובחטי ביצוע מרובים (multithreading) הוא חיוני לביצועים אופטימליים של תוכנה מודרנית. הבנת השימוש הנכון בכלים אלו היא קריטית:

- **Multiprocessing:** מתאים לפעולות תובעניות מבחינת מעבד (CPU-bound)
- **Multithreading:** מתאים לפעולות תובעניות מבחינת קלט/פלט (I/O-bound)

### 2.11 רשימת בדיקה: Multiprocessing

#### 1. זיהוי פעולות מתאימות:

- □ האם זיהיתם פעולות במערכת שתובעניות מבחינת מעבד?
- □ האם פעולות אלו מתאימות למקבול (parallelization)?
- □ האם הערכתם את התועלת הפוטנציאלית מביצוע מקבילי?

#### 2. יישום multiprocessing:

- □ האם הקוד משתמש במודול multiprocessing של Python?
- □ האם מספר התהליכים (processes) מוגדר באופן דינמי על פי מספר הליבות?
- □ האם קיימת טיפול תקין בשיתוף נתונים בין תהליכים?

#### 3. ניהול משאבים:

- □ האם התהליכים נסגרים כראוי בסיום העבודה?
- □ האם קיימת טיפול בחריגות (exceptions) בתוך תהליכים מקבילים?
- □ האם נמנעים מזליגת זיכרון (memory leaks)?

### 3.11 רשימת בדיקה: Multithreading

#### 1. זיהוי פעולות מתאימות:

- □ האם זיהיתם פעולות במערכת שתובעניות מבחינת קלט/פלט?
- □ האם פעולות אלו כוללות המתנה (למשל, קריאות רשת או דיסק)?
- □ האם הערכתם את התועלת הפוטנציאלית מביצוע במקביל?

#### 2. יישום multithreading:

- □ האם הקוד משתמש במודול threading של Python?
- □ האם חטי הביצוע (threads) מנוהלים בצורה מסודרת?
- □ האם קיימת סנכרון נכון בין חוטים (למשל, locks, semaphores)?

#### 3. בטיחות חוטים (thread safety):

- ☐ ☐ האם נמנעים ממצבי תחרות (race conditions)?
- ☐ ☐ האם משתנים משותפים מוגנים באמצעות מנעולים (locks)?
- ☐ ☐ האם נמנעים מנעילה הדדית (deadlocks)?

#### 4.11 שיקולים נוספים

- ☐ האם שקלתם שימוש ב-asyncio לפעולות אסינכרוניות במקום threading?
- ☐ האם בחרתם את הכלי הנכון (תהליכים לעומת חוטים) לכל משימה?
- ☐ האם ביצעתם מדידות ביצועים (benchmarks) לאימות השיפור?

#### 5.11 הערות נוספות

רשמו כאן הערות על ממצאים, בעיות שזוהו, או שיפורים נדרשים:

## 21 בדיקה ג: עיצוב מבוסס אבני בניין

### 1.21 רקע

עיצוב מבוסס אבני בניין (building blocks design) הוא גישה מודולרית לארכיטקטורת תוכנה. כל אבן בניה היא יחידה עצמאית עם:

- **נתוני קלט (input data):** המידע הנדרש לביצוע הפעולה
- **נתוני פלט (output data):** התוצרים שהאבן מייצרת
- **נתוני הגדרה (setup data):** פרמטרים וקונפיגורציה לאבן הבניה

### 2.21 עקרונות עיצוב

עיצוב טוב של אבני בניין צריך לעמוד בעקרונות הבאים:

1. **אחריות יחידה (Single Responsibility):** כל אבן בניה אחראית למשימה אחת מוגדרת
2. **הפרדת דאגות (Separation of Concerns):** כל אבן בניה עוסקת בהיבט אחד של המערכת
3. **קלות שימוש חוזר (Reusability):** אבני הבניה ניתנות לשימוש חוזר בהקשרים שונים
4. **יכולת בדיקה (Testability):** כל אבן בניה ניתנת לבדיקה באופן עצמאי

### 3.21 רשימת בדיקה: זיהוי אבני בניין

#### 1. מיפוי המערכת:

- □ האם יצרתם תרשים זרימה של המערכת שלכם?
- □ האם זיהיתם את כל אבני הבניה העיקריות?
- □ האם מיפיתם את הקשרים והתלויות בין אבני הבניה?

#### 2. הגדרת אבני בניה:

- □ האם כל אבן בניה מוגדרת כמחלקה (class) או פונקציה נפרדת?
- □ האם לכל אבן בניה יש שם תיאורי וברור?
- □ האם לכל אבן בניה יש תיעוד (docstring) מפורט?

### 4.21 רשימת בדיקה: נתוני קלט (Input Data)

בדקו את נתוני הקלט לכל אבן בניה במערכת שלכם:

#### 1. הגדרה ברורה:

- □ האם כל נתוני הקלט מתועדים בצורה ברורה?
- □ האם סוגי הנתונים (data types) מפורטים?

☐ ☐ האם התחום התקף (valid range) לכל פרמטר מוגדר?

## 2. ולידציה (validation):

- ☐ ☐ האם קיימת בדיקת תקינות לכל נתוני הקלט?
- ☐ ☐ האם הבדיקות מטפלות בקלטים שגויים בצורה ראויה?
- ☐ ☐ האם מוחזרות הודעות שגיאה ברורות למשתמש?

## 3. תלויות:

- ☐ ☐ האם כל התלויות החיצוניות מזוהות?
- ☐ ☐ האם התלויות מסופקות באמצעות הזרקת תלות (dependency injection)?
- ☐ ☐ האם אבן הבניה אינה תלויה בקוד ספציפי למערכת?

## 5.21 רשימת בדיקה: נתוני פלט (Output Data)

בדקו את נתוני הפלט לכל אבן בניה במערכת שלכם:

### 1. הגדרה ברורה:

- ☐ ☐ האם כל נתוני הפלט מתועדים בצורה ברורה?
- ☐ ☐ האם סוגי הנתונים (data types) מפורטים?
- ☐ ☐ האם פורמט הפלט עקבי ומוגדר היטב?

### 2. עקביות:

- ☐ ☐ האם הפלט תואם את ההגדרה בכל מצב?
- ☐ ☐ האם מטופלים כל מצבי הקצה (edge cases)?
- ☐ ☐ האם הפלט זהה בכל הרצה עם אותם קלטים (בהנחה שאין אקראיות)?

### 3. טיפול בשגיאות:

- ☐ ☐ האם פעולות שנכשלו מחזירות הודעת שגיאה ברורה?
- ☐ ☐ האם הפלט במקרה של שגיאה מובחן מפלט תקין?
- ☐ ☐ האם השגיאות מתועדות (logged) כראוי?

## 6.21 רשימת בדיקה: נתוני הגדרה (Setup Data)

בדקו את נתוני ההגדרה לכל אבן בניה במערכת שלכם:

### 1. פרמטרים קונפיגורביליים:

- ☐ ☐ האם כל הפרמטרים הניתנים לקונפיגורציה זוהו?
- ☐ ☐ האם לכל פרמטר יש ערך ברירת מחדל סביר?
- ☐ ☐ האם הפרמטרים נטענים מקובץ קונפיגורציה או משתני סביבה?

### 2. הפרדת קונפיגורציה:

- □ האם קונפיגורציה מופרדת מקוד?
- □ האם ניתן לשנות קונפיגורציה ללא שינוי קוד?
- □ האם קיימות קונפיגורציות שונות לסביבות שונות (פיתוח, בדיקה, ייצור)?

### 3. אתחול (initialization):

- □ האם אבן הבניה מאותחלת כראוי לפני שימוש?
- □ האם קיימת פונקציית setup או initialize ייעודית?
- □ האם האתחול מטופל בחריגות אפשריות?

## 7.21 דוגמה לאבן בניה טובה

דוגמה קונספטואלית:

```
class DataProcessor:
    """
    Building block for processing data

    Input Data:
    - raw_data: List of dictionaries
    - filter_criteria: Dict with filtering rules

    Output Data:
    - processed_data: List of processed dictionaries

    Setup Data:
    - processing_mode: str ('fast' or 'accurate')
    - batch_size: int (default: 100)
    """

    def __init__(self, processing_mode='fast', batch_size=100):
        # Setup/configuration
        self.processing_mode = processing_mode
        self.batch_size = batch_size

    def process(self, raw_data, filter_criteria):
        # Input validation
        # Processing logic
        # Return output
        pass
```

## 8.21 הערות נוספות

רשמו כאן הערות על ממצאים, בעיות שזוהו, או שיפורים נדרשים:

### חלק III

## סיכום והמלצות

### 31 ציון כללי

לאחר שעברתם על כל הבדיקות -- הן האקדמיות והן הטכניות -- חשבו ציון כללי לפרויקט שלכם:

#### 1.31 ציון אקדמי (מהחלק הראשון)

□ ציון משוקלל אקדמי: \_\_\_\_\_/001

#### 2.31 ציון טכני (מהחלק השני)

בדקו כמה פריטים מהבדיקות הטכניות עברו:

□ סך הפריטים הטכניים שעברו: \_\_\_\_\_

□ סך כל הפריטים הטכניים: \_\_\_\_\_

□ אחוז הצלחה טכני: \_\_\_\_\_%

#### 3.31 ציון כולל

המלצה לשקלול:

□ ציון אקדמי: 60%

□ ציון טכני: 40%

□ ציון כולל סופי: \_\_\_\_\_/100

### 41 תחומים לשיפור

רשמו את שלושת התחומים העיקריים הדורשים שיפור בפרויקט שלכם:

1. \_\_\_\_\_

2. \_\_\_\_\_

3. \_\_\_\_\_

## 51 תכנית פעולה

לכל תחום שיפור, הגדירו צעדי פעולה קונקרטיים:

## 61 לסיכום

ההערכה העצמית היא תהליך מתמשך שמשלב הן רפלקציה אקדמית והן בדיקה טכנית מעמיקה. חזרו על תהליך זה באופן קבוע (למשל, כל חודש או בסוף כל שלב פיתוח) כדי לוודא שהפרויקט שלכם ממשיך להשתפר ולעמוד בסטנדרטים הגבוהים ביותר -- הן מבחינת התיעוד והמחקר, והן מבחינת איכות הקוד והעיצוב הטכני.

**בהצלחה! זכרו: הערכה עצמית כנה ומדויקת היא סימן לבגרות אקדמית ומקצועיות.**

---

2025 Dr. Yoram Segal. All Rights Reserved. □

Version 2.0 -- 22-11-2025