עבודה 1

חלק 1:

1. א. פרדיגמה זו היא פרדיגמת תכנות המשתמשת בהצהרות המשנות את מצב התוכנית. תוכנית אימפרטיבית מורכבת מרצף של פקודות לביצוע כאשר הפקודות מבוצעות בזו אחר זו. מתמקדת בתיאור אופן הפעולה של תוכנית.
ב. פרדיגמה זו היא פרדיגמת תכנות הנגזרת מתכנות אימפרטיבי ומבני, המבוססת על הרעיון של *procedure call*. יש בה סדרה של צעדים חישוביים שיש לבצע אשר יכולים לבוא לידי ביטוי במבנים מורכבים יותר כמו לולאות ופונקציות. תחביר השפה מאפשר לכתוב קוד כפרוצדורה כך שניתן לקרוא לה ממוקמות שונים בקוד.
ג. פרדיגמה זו היא פרדיגמת תכנות בה מנסים לאגד הכל בסגנון פונקציות מתמטיות טהורות (זהו סוג הצהרתי של סגנון תכנות). ההתמקדות העיקרית שלה היא במה לפתור. היא משתמשת בביטויים במקום בהצהרות. הרצת תוכנית בפרדיגמה זו לא מתבצעת על ידי רצף של פקודות, אלא על ידי חישוב ביטויים (מציאת הערך שלהם) כאשר לא מתבצעות פעולות השמה, כלומר אין side-effect. בפרדיגמה זו גם פונקציה היא ביטוי, לכן בין היתר ניתן גם להעביר פונקציה כפרמטר לפרוצדורה אחרת, וגם להחזיר כערך מפרוצדורה פונקציה, מה שמכונה כפונקציות מסדר גבוה.

- <u>כיצד משתפרת הפרדיגמה הפרוצדורלית על פני הפרדיגמה האימפרטיבית?</u>
הפרדיגמה האימפרטיבית מבוססת על רצף של פקודות בזו אחר זו, דבר זה עלול לגרום לחזרתיות בקוד שאינה לצורך, דבר הגורם לקוד להיות פחות קריא ומובן וגם עלול להוות מתכון לבאגים שונים בקוד (אם כותבים משהו מספר פעמים, עולה ההסתברות שבאחת הפעמים נכתוב משהו בצורה לא תקינה). לעומת זאת, הפרדיגמה הפרוצדורלית אשר גם מבוססת על הפרדיגמה המבנית, מכילה לולאות, מבני נתונים מורכבים יותר ומשתנים אשר מונעים חזרה מיותרת של הקוד וגורמים לכך שהקוד יהיה מובן יותר בעיניי הקורא. בנוסף, בפרדיגמה הפרוצדורלית כאשר אנו כותבים פונקציה מסוימת, הפונקציה תלויה בקלט שיגיע מהמשתמש ובכך (בניגוד לפרדיגמה האימפרטיבית) לא צריך לשנות את הקוד כאשר רוצים לשנות את הקלט לפעולה מסוימת בתוכנית אשר מבוטאת בפרדיגמה הפרוצדורלית כפונקציה.
יתרון נוסף בפרדיגמה הפרוצדורלית הוא שכאשר אנו כותבים פונקציה מסוימת, ניתן לקרוא לפונקציה מספר פעמים במהלך הקוד ובנוסף, מתבצעת הפרדה בין פונקציה זו לבין יתר הקוד שבתוכנית ובכך יש הפרדה בין הלוגיקה של הפונקציה לבין יתר הקוד שבתוכנית. דבר זה, כפי שצוין מקודם, מעניק הבנה טובה יותר לקוד וגם גורם לכך שהפונקציה שנכתבה מוכנה לפעולה ללא תלות בכתיבת יתר הקוד שבתוכנית.
- <u>כיצד משתפרת הפרדיגמה הפונקציונאלית לעומת הפרדיגמה הפרוצדורלית?</u>
התכנות הפונקציונלי משפר בין היתר את הדברים הבאים:
<u>אימות של הקוד</u>- מכיוון שבתכנות פונקציונלי אין side-effects, כל פונקציה למעשה רק מקבלת פרמטרים מסוימים ומחזירה תשובה כשלהי וזאת מבלי לשנות את הזיכרון, ומבלי לשנות דברים בפונקציות אחרות או בשאר הקוד של התוכנית באופן כללי, ולכן ניתן לאמת כל פונקציה בנפרד ביותר קלות מאשר בפרדיגמה הפרוצדורלית.
<u>מקבול</u>- מכיוון שבתכנות פונקציונלי לא מתבצעת כתיבה לזיכרון (אין השמות) אלא רק קריאה מהזיכרון, קל יותר למתכנת לעצב תוכנית מקבילית בלי צורך להשתמש במנגנוני סנכרון כלשהם, זאת בניגוד לפרדיגמה הפרוצדורלית, שם ייתכן שתהיה כתיבה מקבילית לזיכרון.
<u>הפשטה / עיצוב</u>- כפי שצוין בהסבר על הפרדיגמה הפונקציונלית, ניתן בתכנות זה להשתמש בפונקציות מסדר גבוה ובכך ניתן להפוך את הקוד למובן יותר, לעומת הפרדיגמה הפרוצדורלית שבה הלוגיקה מורכבת מדי שזה מקשה על הבנת הקוד.

2. א. $(x, y) \Rightarrow x.some(y)$

$< T1 > (x: T1[], y: (z: T1) \Rightarrow boolean) \Rightarrow boolean$

ב. $\mathrm{x} \Rightarrow \mathrm{x.reduce((acc, cur) \Rightarrow acc + cur, 0)}$

$\mathrm{(x: number[]) \Rightarrow number}$

ג. $\mathrm{(x, y) \Rightarrow x\ ?\ y[0]\ :\ y[1]}$

$< T1 > \mathrm{(x: boolean, y: T1[]) \Rightarrow T1}$

3. *הפשטה* –תהליך שבאמצעותו נתונים והוראות מוגדרים ע"י הצגה דומה למשמעות הסמנטית שלהם תוך הסתרת פרטי המימוש שלהם. מטרתה להציג רק את הפרטים הרלוונטיים לנקודת המבט הנוכחית ע"י הקטנת כמות הפרטים ו"הוצאה החוצה" של הטיפול בהם, כך שהמתכנת יוכל להתמקד רק בכמות מוגבלת של רעיונות בפרק זמן נתון.
מחסום ההפשטה – מחלק את העולם לשני חלקים: לקוחות המשתמשים בתוכנה, ומיישמים שבונים אותה (מתכנתים). הלקוחות לא צריכים לדעת איך התוכנה עובדת, כדי לבצע את עבודתם, יש להם סט מוגבל של פעולות בהם הם יכולים להשתמש על מנת לעצב את התוכנית, ומעבר לכך עליהם לסמוך על התוכנה. המתכנת צריך לדעת כיצד התוכנה עובדת, ובמקרה של תקלות בתוכנה עליו לדעת כיצד לפתור אותן. נוסיף שהמתכנת הוא גם לקוח כי הוא משתמש גם בתוכנות אחרות ליישום התוכנה שהוא בונה.