# עקרונות שפות תכנות - עבודה 1

**שאלה 1**

1. Domain – לאילו סוגי אפליקציות מתאימות יותר שפות השייכות לפרדיגמה מסוימת. למשל:
   1. לתשאול DBs מתאימות יותר שפות השייכות ל Declarative paradigm כדוגמת SQL, כי לא בהכרח מעניין מימוש השליפה, אלא בעיקר התוצאה שלה.
   2. לכתיבה של רכיבי חומרה יותר מתאימות שפות השייכות ל Imperative paradigm כדוגמת אסמבלי, basic וכו'.
   3. עבור כתיבת תוכנה שנדרש להוכיח את הנכונות שלה מתמטית, נעדיף להשתמש בשפה פונקציונלית כי היא מאפשרת לבצע זאת באופן הקל והנוח ביותר.
2. Coupling and reuse – באיזו מידה ניתן לבצע reuse לקוד בחלקים שונים של התוכנית. למשל:
   1. בשפות השייכות ל Structured paradigm (כמו C) נשתמש בלולאות ובפונקציות, המאפשרות לחלק את הקוד ליחידות קטנות המבצעות פעולות בסיסיות, ולהשתמש בהן לאורך התוכנית בנוחות.
   2. בשפות אימפרטיביות, קשה לחלק את הקוד ליחידות קטנות (ניתן לבצע קפיצות למקומות שונים) דבר שמקשה לעשות reuse לקוד.
   3. בתכנות פונקציונלי קל לעשות שימוש חוזר בקוד על ידי קריאה לפונקציות.
3. Syntax – כמה נוח וטבעי לקרוא קוד שנכתב על ידי שפה מסוימת, ומה מידת החופש של המתכנת בהרחבת השפה. למשל:
   1. בשפה אימפרטיבית, כמו אסמבלי, קשה מאד להרחיב את השפה וכן קשה מאד להבין מה מבצעת תוכנית מורכבת.
   2. בשפה השייכת ל Structured paradigm, קל יותר לקרוא את הקוד בזכות מבנים בשפה כמו לולאות, switch-case, if-else, מבנים שצורת הכתיבה שלהם גם מזכירה שפה אנושית.
   3. בתכנות פונקציונלי, בדרך כלל הקוד יהיה קצר יותר (למשל שימוש בקריאה לmap במקום בלולאה) ולכן עשוי להיות קריא יותר, וגם שימוש בפונקציות בעלות שם אינדיקטיבי מזכירה שפה אנושית.

**שאלה 2**

|  |  |
| --- | --- |
| **Expression** | **Type** |
| (x, y) => x + y | (x:number, y:number) => number |
| x => x[0] | <T>(x: T[] | string) => T | string |
| (x, y) => x ? y : -y | (x:boolean ,y:number) => number |

**שאלה 3**

Shortcut semantics הוא אופן הערכה (evaluation) של סדרת ביטויים בוליאניים כך שברגע שנתקלים בביטוי שהערכה שלו קובעת בוודאות את ערך הביטוי הכולל, מיד מחזירים את הערך הזה מבלי להעריך את שאר הביטויים שנותרו.

לדוגמה, בסדרת הביטויים false || 1 === 3 || true || 8 === 8 || true === false  
תת-הביטוי הכחול קובע את ערכו של הביטוי הכולל כ-true ולכן אין צורך להמשיך להעריך את שאר תתי-הביטויים (בכתום).

דוגמה נוספת היא פונקציית every ב-Javascript שמופעלת על מערך ומקבלת פרדיקט, אשר עוברת על איברי המערך (לא בהכרח על כולם) ומחזירה false ברגע שהיא מעבדת איבר שלא מקיים (satisfies) את הפרדיקט. היא תחזיר true רק עם כל האיברים מקיימים את הפרדיקט.