עבודה 1 בקורס עקרונות שפות תכנות
חלק 1

1. א. **אימפרטיבית-** שפה שמורכבת ממספר פקודות בזו אחר זו. בשפה זו אנו מסבירים למחשב איך למצוא או לחשב את התוצאה המבוקשת.

ב. **פרוצדורלית-** תחביר השפה מאפשר להגדיר קוד כפרוצדורה, אשר ניתן לקרוא ממקומות שונים בקוד.

ג. **פונקציונלית-** התוכנית היא ביטוי, או סדרת ביטויים, ולא רצף של פקודות.
הרצת התוכנית היא חישוב של הביטוי(ים), כלומר מציאת הערך שלו, ולא ביצוע של הפקודות.

יתרונות של פרוצדורלי על אימפרטיבי:
- ניתן להשתמש באותו קטע קוד ממספר מקומות שונים בתוכנית.
- הקוד הוא לא ספציפי עבור חישוב מסוים.
- תחביר השפה מלמד על דפוס חוזר בתכנית (מקל על הבנה).

יתרונות של פונקציונלי על אימפרטיבי:
- עוזרת לאימות הקוד ובידוד הפעולות השונות.
- בתכנות פרוצדורלי יש נטייה לשימוש באובייקט משותף, דבר אשר עלול לפגוע באפשרות למקביליות. בניגוד לתכנות פונקציונאלי.

.2

```
const isOverSixty = (x:number) => x > 60;
const devide = (x:number,y:number) => x/y;
function MyaverageGradesOver60(grades: number[]) {
    const arrayOver60 = grades.filter(isOverSixty);
    const arraySum = arrayOver60.reduce((acc, cur) => acc + cur , 0);
    return devide(arraySum,arrayOver60.length);
}
```

.3

```
//(a)
const func1: <T>(x:T[], y: (t:T) => boolean) => boolean = <T>(x:T[], y: (t:T) => boolean): boolean =>{
    return x.some(y);
}

//(b)
const func2: (x: number[]) => number = (x: number[]): number =>{
    return x.reduce((acc: number, cur:number) => acc + cur, 0);
}

//(c)
const func3: (x:boolean, y:any[]) => any = (x:boolean, y:any): any => {
    x ? y[0] : y[1];
}

//(d)
const func4: <T,U,W>(f: (y:T) => U,  g: (x:W) => T) => U = <T,U,W>(f: (y:T) => U,  g: (x:W) => T) : U => {
    return f(g(x+1));
}
```

.4

Abstraction Barriers – The concept behind abstraction barriers is to separate the use of an abstraction from the implementation of it.
Abstraction barriers raise essential complexity and hide accidental complexity.
Meaning the programmer is free to make changes and improvements to his program while not concerning himself with details over underlying implementations of the abstraction he is using, which is effectively hidden beneath what we name the abstraction barrier.
This way all the programmer using this abstraction needs to know is what operations are available to him.