**עקרונות שפות תכנות:**

תרגיל 4:

תאריך הגשה: 10.03.24

הוראות הגשה: ההגשה בזוגות / כיחידים דרך מערכת הסאבמיט. כל זוג נדרש לחשוב, לפתור ולכתוב את התרגיל בעצמו. יש לקרוא הוראות אלא בקפידה. הגשה שלא על פי הוראות אלה תוביל להורדת ניקוד.

מה יש להגיש:

```
ex4_1.js
ex4_2.js
ex4_3.js
rotate.pl
fibo.pl
letter_num_list.pl
```

קובץ עם השם משתמש בסבמיט ות.ז של כל אחד מהמגישים באופן הבא:

id.txt:

```
301111111 NOFRI
209111111 YOSIM
```

כל הקבצים צריכים להיות בקובץ zip אחד בשם: ex4.zip

**חלק א': JavaScript:**

יש לממש את כל התרגילים בקבצים המצורפים ללא שינוי של הקוד למעט המקומות שנכתב בהם במפורש.

The following function `createStack()` creates instances of the stack data structure:

```
function createStack() {
  return {
      items: [],
      push(item) {
      this.items.push(item);
      },
      pop() {
      return this.items.pop();
      }
  };
}

const stack = createStack();
stack.push(10);
stack.push(5);
stack.pop(); // => 5

stack.items; // => [10]
stack.items = [10, 100, 1000]; // Encapsulation broken!
```

The stack works as expected, but with one small problem. Anyone can modify any item directly because `stack.items` property is exposed.

That's an issue since it breaks the encapsulation of the stack: only `push()` and `pop()` methods should be public, but `stack.items` or any other details shouldn't be accessible.

Refactor the above stack implementation, using the concept of closure, such that there is no way to access `items` array outside of `createStack()` function scope:

```
function createStack() {
  // Write your code here...
}

const stack = createStack();
stack.push(10);
stack.push(5);
stack.pop(); // => 5

stack.items; // => undefined
```

2. Write a function `multiply()` that multiplies 2 numbers:

```
function multiply(num1, num2) {
  // Write your code here...
}
```

If `multiply(num1, numb2)` is invoked with 2 arguments, it should return the multiplication of the 2 arguments.

But if invoked with 1 argument `const anotherFunc = multiply(num1)`, the function should return another function. The returned function when called `anotherFunc(num2)` performs the multiplication `num1 * num2`.

```
multiply(4, 5); // => 20
multiply(3, 3); // => 9

const double = multiply(2);
double(5);  // => 10
double(11); // => 22
```

3. Implement currying for a function with multiple parameters using closure:

```
function curry (fn) {
  // Write your code here...
}
function multiply (a, b, c) {
      return a * b * c;
}

var curryMultiply = curry(multiply);
```

```
console.log(curryMultiply(2)(3)(4)); // 24
```

**חלק ב (Prolog):**

חלק זה בשפת prolog, עבור כל סעיף צריך לכתוב קובץ .pl שהשם שלו הוא שם ה"פונקציה" שאותה יש לכתוב. למשל, עבור שאלה 1 יש לכתוב קובץ בשם rotate.pl .

**שאלה 1:**

ממשו את:

```
rotate(L, L1).
```

המקבלת רשימה ומחזירה רשימה המייצגת את הפרמוטציה הבאה בתור.

עבור הרשימה הריקה, או רשימה עם איבר אחד, הרשימה המתקבלת תהיה זהה לרשימה המקורית.

לדוגמה:

```
rotate([2, 3, 4, 5, 6], X].
X = [3, 4, 5, 6, 2]
```

**שאלה 2:**

ממשו את:

```
fibo(N, N1).
```

המקבלת אינדקס N ומחזירה את המספר פיבונאצ'י הnth.

לדוגמה:

```
fibo(7, N1).
N1 = 13.
```

**שאלה 3:**

ממשו את:

```
letter_num_list(L, [X, Y]).
```

המקבלת רשימה L ומחזירה רשימה המפרידה את הרשימה L לשתי רשימות אחת (X) עם אותיות בלבד ואחרת (Y) עם ספרות בלבד.

לדוגמה:

```
letter_num_list([1,4,a,b,2,d],X).
X = [[1,4,2], [a,b,d]].
```