עקרונות שפות תכנות:

תרגיל 1:

תאריך הגשה: 21.1.24

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.

סקריפט בדיקה: מצורף סקריפט לבדיקה, מומלץ לבדוק את התרגיל בעזרתו טרם ההגשה.

כמובן שאריץ בדיקות נוספות.

נא וודאו שהריצה של הסקריפט עובדת על הקוד.

קבצים להגשה:

- lists.ml
- trees.ml
- variants.ml
- hardquestions.ml
- ex1.pdf (עם הפתרונות של החלק התיאורטי.)

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: ex1.zip

חלק א': OCAML

1. עבודה עם רשימות:

1. ממשו את הפונקציה `sum_list`.

קלט: רשימה (ניתן להניח כי הרשימה מכילה רק שלמים)

פלט: סכום האיברים ברשימה.

דוגמאות ריצה:

```
sum_list [1;2;3;4;5] ;;
- : int = 15
```

2. ממשו את הפונקציה `compress`

קלט: רשימה.

פלט: רשימה ללא כפילויות של איברים עוקבים.

```
compress [1; 1; 2; 3; 3; 4; 4; 4; 5] ;;
[- : int list = [1; 2; 3; 4; 5
```

2. עבודה עם עצים:

נתונה ההגדרה לעץ בינארי:

```
# type 'a binary_tree =
    | Empty
    | Node of 'a * 'a binary_tree * 'a binary_tree ;;
type 'a binary_tree = Empty | Node of 'a * 'a binary_tree * 'a binary_tree
```

1. כתבו את הפונקציה: `construct`

קלט: רשימה של מספרים (int).

פלט: עץ חיפוש בינארי.

דוגמאות ריצה:

```
# construct [4; 1; 5; 7; 3] ;;
- : int binary_tree =
Node (4, Node (1, Empty, Node (3, Empty, Empty)),
 Node (5, Empty, Node (7, Empty, Empty)))
```

ב. מה יש לשנות בפונקציה מסעיף א כדי שתתמוך גם בעץ חיפוש בינארי
שכל איבריו הם מספרים של נקודה צפה (floats)?

הערה: תשובה לסעיף ב צריכה להתווסף לקובץ התשובות pdf.

3. **<u>וריאנט:</u>**

א. הגדירו טיפוס חדש באוקמל בשם shape, עם שלושה בנאים:
- Circle עם ארגומנט המייצג את הרדיוס שלו (float).
- Square עם ארגומנט המייצג אורך צלע (float).
- Rectangle עם שני ארגומנטים המייצגים את האורך והגובה (float).

ב. ממשו את הפונקציה `area`.
קלט: ארגומנט מסוג shape (כפי שהגדרתם מלעיל)
פלט: שטח הצורה.

```
area (Circle 5.0) ;;
- : float = 78.53975
```

הערה: `pi = 3.14159`

ג. ממשו את הפונקציה: `total_area`
קלט: רשימה של צורות.
פלט: סכום השטחים של הצורות במערך.

<u>3. שאלות מאתגרות:</u>

1. ממשו את הפונקציה: `arithmetic_hell (כבר לא כזה hell)`
קלט: רשימה של מספרים שלמים (int).
<u>פלט:</u> כמות המשוואות החוקיות אריתמטית של המספרים ברשימה, ללא שינוי סדר האיברים, כאשר הפעולות האפשריות הן +, - וסימן השיוויון באיבר האחרון בלבד.

הסבר מפורט:

קיבלנו את הרשימה הבאה:

```
[2;2;1;1;2]
```

אפשר לבחור להכניס כל אופרטור מתמטי: `+ -` בין כל זוג איברים עוקבים ברשימה.
כמו כן, אפשר להכניס את סימן השוויון `=` **רק בין האיבר האחרון לאחד לפניו**
סה"כ עלינו ליצור משוואה אריתמטית חוקית.

בדוגמה זו קיימות 2 קומבינציות אפשריות עבור רשימת המספרים למשוואות נכונות אריתמטית. המשוואות לצורך הדוגמה הן:

$$2 - 2 + 1 + 1 = 2$$
$$2 + 2 - 1 - 1 = 2$$

אין צורך להחזיר את המשוואות עצמן, רק את מספר המשוואות החוקיות.

רמז: זה תרגיל קשה (כבר לא כזה קשה), שימו לב, תנסו לחשוב איך עושים את זה עם in (למדנו) ובעזרת accumulator.

חלק ב': אינדוקציה מבנית:

שאלה 1:

נגדיר:

```
type element =
  | Z1
  | Z2 ;;

type sequence =
  | Empty
  | Cons of element * sequence ;;

let rec len = function
  | Empty -> 0
  | Cons (_, tail) -> 1 + len tail ;;

let rec append x y =
  match x with
  | Empty -> y
  | Cons (head, tail) -> Cons (head, append tail y) ;;
```

<u>הוכיחו כי:</u>

$$\forall x, y \in sequence\colon\ len(append\ x\ \ y) = len(x) + len(y)$$

<u>דוגמת הרצה:</u>

```
(* EXAMPLE: *)
let seq1 = Cons(Z1, Cons(Z2, Empty));;  (* Sequence: Z1 -> Z2 -> Empty *)
let seq2 = Cons(Z2, Cons(Z1, Empty));;  (* Sequence: Z2 -> Z1 -> Empty *)

(* Concatenate the sequences *)
let combined_seq = append seq1 seq2;;  (* Result: Z1 -> Z2 -> Z2 -> Z1 -> Empty *)

(* Calculate the lengths of the sequences *)
let len_seq1 = len seq1;;  (* Expected length: 2 *)
let len_seq2 = len seq2;;  (* Expected length: 2 *)
let len_combined_seq = len combined_seq;;  (* Expected length: 4 *)
```

שאלה 3:

נתונה הגדרה לעץ בינארי פשוט:

```
type 'a btree =
      | Empty
      | Node of 'a * 'a btree * 'a btree
```

כמו כן, נתונה הפונקציה הבאה:

```
let rec height = function
      | Empty -> 0
      | Node(_, left, right) -> 1 + max (height left) (height right)
```

הוכיחו בעזרת אינדוקציה מבנית:

לכל עץ בינארי t מסוג btree מתקיים כי height t גדול או שווה מאורך המסלול הארוך ביותר בין שורש העץ לאחד העלים שלו.

שאלה 4:

נגדיר:

.

```
type bool_expr =
  |    Var of string
  |    Not of bool_expr
  |    And of bool_expr * bool_expr
  | Or of   bool_expr * bool_expr;;
```

נכתוב את הפונקציות הבאות:

```
let rec num_of_vars = fun exp -> match exp with
| Var(_) -> 1
| And(x,y) -> (num_of_vars x) + (num_of_vars y)
| Or (x,y) -> (num_of_vars x) + (num_of_vars y)
| Not(x) -> (num_of_vars x);;

let rec num_of_connectives = fun exp -> match exp with

| Var(_) -> 0
| And(x,y) -> (num_of_connectives x) + (num_of_connectives y) + 1
| Or(x,y) -> (num_of_connectives x) + (num_of_connectives y) + 1
| Not(x) -> (num_of_connectives x) + 1;;
```

א. הוכיחו באינדוקציה מבנית או הפריכו באמצעות דוגמא נגדית: לכל ביטוי exp מטיפוס

`bool_expr` מתקיים:

```
num_of_vars(exp) = num_of_connectives(exp) + 1
```

ב. הוכיחו באינדוקציה מבנית או הפריכו באמצעות דוגמא נגדית: לכל ביטוי exp מטיפוס

`bool_expr` בו לא מופיע Not מתקיים:

```
num_of_vars(exp) = num_of_connectives(exp) + 1
```