

חלק יבש

שאלה 1:

עבור כל אחת מהשפות הבאות קבע לאילו פרדיגמות היא שייכת והסבר בקצרה.
scala, rust, ruby, javascript, clojure, SPARQL, ATS

Scala: functional, imperative

שפה מונחית עצמים כי כל ערך בה הוא עצם

שפה פונקציונלית כי כל פונקציה היא ערך ולכן אפשר להשתמש בה כמו ערך.

Rust: concurrent, functional, generic, imperative, structured

שפה שיש לה ביצועיות גבוהה ושמה דגש על בטיחות ומהירות.

Ruby: functional, imperative, object-oriented, reflective

שפת מונחת עצמים טהורה. יש בה ביטויים רגלוריים. יש לה garbage collector.

Javascript: functional, imperative

שפת תכנות מתאימה לאתרי אינטרנט ורצה על ידי דפדפן האינטרנט בצד הלקוח.

השפה משתמשת במפרש. השפה תומכת בסינטקס של שפת C אך היא גם סוג של מונחת עצמים. השפה היא מונחת אירועים.

Clojure: concurrent, functional

השפה מעודדת תכנות פונקציונלי ו"מקומפלט" ע"י על JVM.

SPARQL: Query language

שפת שאלות למסדי נתונים. פקודות בשפה זו נועדו לטפל בנתונים.

ATS: functional, imperative

שפת תכנות שנועדה לאחד תכנות עם מפרט רשמי. מערכת הטיפוסים של ATS מאפשרת תכנות רבות שנחשבות "לא בטוחות" (למשל, הקצאת זיכרון), מה שהופך את ATS לשפת תכנות עם בטיחות ברמה נמוכה.

שאלה 2:

tail-recursion (רקורסיית זנב)

1. הסבר מה זה tail-recursion.

רקורסיית זנב היא פונקציה רקורסיבית שהיא מתכוננת, כך שהפקודה האחרונה של הפונקציה, זוהי הקריאה לפונקציה עצמה, ובעצם החישוב של הביטוי מסתיים בקריאה האחרונה לפונקציה. ז"א הפונקציה לא מתקפלת חזרה בעץ הרקורסיה אלא פשוט מחזירה ערך (בתחתית העץ).

2. איך ניתן להשתמש ב-tail-recursion כדי ליעל תוכניות בשפות פונקציונליות? ניתן להשתמש בהם על ידי כך שנעשה את החישובי הנדרשים לפני הקריאה ונכוון את הביטוי כך שהפקודה האחרונה בפונקציה כזו, תהיה הקריאה לפונקציה עצמה, ובכך נחסוך בסיבוכיות מקום וזמן.

3. כתוב ב-ML פונקציה שהיא tail-recursive המקבלת מספר טבעי n ומחזירה את n!.

```
fun fact n =let
  fun hlpfact 0 sum = sum
    | hlpfact n sum = hlpfact(n-1,n*sum)
in
  hlpfact n 1
end;
```

שאלה 3:

1. כתבו בקצרה את ההבדלים העיקריים בין BNF ל-EBNF.

זה כללים בסיסיים להגדרת שפות-BNF

זה הרחבה של BNF (Syntactic Sugar) מאפשר-EBNF

2. כתבו EBNF קצר המגדיר את ליטרל המספרים: מספרים שלמים (למשל, 17), מספרים עם נקודה עשרונית (למשל, 63.77) ומספרים בכתוב מדעי (למשל, E-81.5). הן חיוביים והן שליליים.

number= [[sign], digit],{digit}, ["."], {digit}, [exp];
digit= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
sign= "+" | "-";
exp= "E", [sign], digit ,{digit}, ["."], {digit};

3. כעת כתבו regular expression המגדיר את אותו הליטרל. ביטוי רגולרי ללא הסבר מספק לא יתקבל.

הערה: מטרתכם היא לתפוס את כל הליטרל, ללא תלות בערך שהוא מייצג. לכן גם

הליטרלים: 00.3400, -0.0, 014, -3, 7, 1.3+ הם כולם ליטרלים חוקיים.

מומלץ להשתמש במנוע www.regexr.com לבדיקת נכונות הביטוי.

([-+]?[0-9]*([-+]?[0-9]+)?[0-9]*([E]([[-+]?[0-9]+))?[0-9]*([-+]?[0-9]+)?[0-9]*)

1 אז בהתחלה שמנו את "+" או "-" כאופציה כי זה לא הכרחי =[-+]?([-+]?[0-9]+)?

2 לאחר מכן אפשרות לשים ספרות ככל שרוצים עם נקודה ביניהם עם מספר אחד לפחות

$[0-9]^+([0-9].)^*[0-9]^*$ אחרי הנקודה

3 אפשרות בחירה לשים E ואופציה לסימן אח"כ אבל חייב לשים ספרה אחת לפחות

$[0-9]^+([E]([+-])?)^*[0-9]^*$ אחריהם

4 ושוב כמה ספרות שרוצים עם נקודה $[0-9]^+([0-9].)^*[0-9]^*$

שאלה 4:

ב-ML ניתן להגדיר פונקציה "בתוך" פונקציה אחרת, למשל:

```
fun foo () =  
  let  
    fun bar x = 2 * x  
  in  
    bar  
  end;
```

דבר זה אפשרי גם ב-Rust, למשל:

```
fn foo() -> fn(i32)->i32 {  
  fn bar(x: i32) -> i32 { 2 * x }  
  bar  
}
```

1. מצא הבדל בין השפות בהקשר זה. (מה המגבלות על פונקציות פנימיות בכל שפה?)

ב-Rust חייבים לציין את סוג הדברים המסוימים ולא קיימת לא מערכת הסקה אוטומטית כמו ML.

2. הבא קטעי קוד המדגימים את ההבדל.