Test PP Nume ..... Grupa .....

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a																				
b																				
c																				
d																				

1. Ce puteți spune despre funcția `len`, în Scheme, și despre aplicația `(len '(1 2 3) 0)`?

```
(define (len L acc)
  (if (null? L)
      acc
      (+ 1 (len (cdr L) acc))))
```

   (a) Funcția este recursivă pe stivă, iar aplicația se evaluează la `3`.
   (b) Funcția este recursivă pe coadă, iar aplicația se evaluează la `3`.
   (c) Funcția este recursivă pe stivă, iar aplicația se evaluează la `0`.
   (d) Funcția este recursivă pe coadă, iar aplicația se evaluează la `0`.

2. Care dintre următoarele este o implementare corectă a funcționalei `map`, în Scheme?

   (a)
```
(define (mymap f L)
  (foldl (lambda (x y) (append y (list (f x))))
         null
         L))
```
   (b) `(define (mymap f L) (apply f L))`
   (c)
```
(define (mymap f L)
  (if (null? L)
      null
      (append (mymap (cdr L)) (list (f (car L))))))
```
   (d) `(define (mymap f L) (list (lambda (x) (f x)) L))`

3. La ce se va evalua următoarea expresie în Scheme?

```
(map (lambda (x) (> x 2)) '(1 2 3 4 5))
```

   (a) `(#f #f #t #t #t)`
   (b) `(3 4 5)`
   (c) `#f`
   (d) `(1 2)`

4. Ce va afișa următorul program în Scheme?

```
(define curry* (lambda (x) (lambda (y) (* x y))))
(define a 10)
(define *a (curry* a))
(*a 5)
(define a 20)
(*a 5)
```

   (a) `50 50`
   (b) `50 100`
   (c) Eroare: `application: not a procedure`
   (d) Eroare: `arity mismatch`

5. Ce rezultat va întoarce următorul program Scheme?

```
(define x 10)
(define pr (lambda (y) (delay (+ x y))))
(+ (force (pr 2)) (force (pr 3)) (force (pr 5)))
```

   (a) `40`
   (b) `36`
   (c) `10`
   (d) Eroare

6. Care dintre următoarele afirmații este adevarată?

   (a) Haskell are tipare statică și utilizează inferență de tip.
   (b) Haskell are tipare dinamică și utilizează inferență de tip.
   (c) Haskell are tipare dinamică și nu utilizează inferență de tip.
   (d) Haskell are tipare statică și nu utilizează inferență de tip.

7. Ce tip are următoarea funcție în Haskell?

```
f x y z = fst (z, [x, y])
```

   (a) `a -> a -> b -> b`
   (b) `a -> b -> c -> c`
   (c) `a -> a -> a -> a`
   (d) `a -> b -> a -> b`

8. Ce tip are funcția `f` în Haskell?

```
f x y z = [x ++ (y : z)]
```

   (a) `[a] -> a -> [a] -> [[a]]`
   (b) `a -> a -> [a] -> [a]`
   (c) `[a] -> [a] -> [a] -> [a]`
   (d) `[a] -> a -> a -> a`

9. Fie următoarele definiții de funcții în Haskell:

```
f x y = x + y
g = \x -> \y -> x + y
```

   Ce întoarce apelul `f == g`?

   (a) Eroare
   (b) `True`
   (c) `False`
   (d) Ciclează

10. Se consideră funcția identitate din Haskell:

```
id :: a -> a
id x = x
```

    Ce vor afișa următoarele apeluri:

```
(id 3) + (length (id "gigi"))
map id ["shaorma", 1, "mai"]
```

    (a) `7` și Eroare
    (b) `7` și `["shaorma", 1, "mai"]`
    (c) Eroare și Eroare
    (d) Eroare și `["shaorma", 1, "mai"]`

11. Care din următoarele expresii Haskell generează lista numerelor impare de la 1 la 10?

    (a) `[ x | x <- [1 .. 10], not $ even x ]`
    (b) `[ odd x | x <- [1 .. 10] ]`
    (c) `[ x | x <- [1 ..], x < 11, x `mod` 2 == 1 ]`
    (d) `[ x | x > 0, x < 11, odd x ]`

12. Care este diferența dintre:

```
p :- a, b.
p :- c.
```

    și

```
p :- a, !, b.
p :- c.
```

    (a) Primul este $p = (a \wedge b) \vee c$, al doilea este $p = (a \wedge b) \vee (\neg a \wedge c)$.
    (b) Niciuna.
    (c) Primul este $p = a \wedge b \wedge c$, al doilea este $p = a \wedge (b \vee c)$.
    (d) Primul este $p = (a \wedge b) \vee c$, al doilea este $p = (a \wedge b) \vee (b \wedge c)$.

13. Ce va afișa interogarea `append([1], X, [2, 3])` în Prolog?

    (a) `false`
    (b) `X = []`
    (c) `X = [1, 2, 3]`
    (d) `Error: Arguments not sufficiently instantiated`

14. Considerând următoarele fapte Prolog:

```
p1(1).
p1(2).
p1(3).
p1(4).
p2(2).
p2(3).
```

ce va afișa interogarea

```
p1(X), p2(Y), write(X), write(Y), X >= Y, !, fail.
```

(exceptând `false`)?

(a) `121322`
(b) `1122`
(c) `12322`
(d) `123223323423`

15. Predicatul `=/2` din Prolog realizează ... între cele două argumente.

(a) unificarea
(b) echivalența
(c) idempotența
(d) unificarea și echivalența simultan

16. Câte soluții va avea următoarea interogare în Prolog:

```
member(a,X).
```

(a) infinit
(b) zero
(c) una
(d) eroare

17. Câte activări va produce *pattern*-ul `(f $?x $?)` cu faptul `(f 1 2)`?

(a) 3
(b) 4
(c) 2
(d) 1

18. Care afirmație este adevarată despre următorul program CLIPS?

```
(deffacts fapt
    (f 1 2 3))
(defrule regula
    ?f <- (f $?)
  =>
    (retract ?f)
    (assert (f 1 2 3)))
```

(a) programul va cicla la nesfârșit
(b) se va produce o singură activare și în baza de fapte vom avea `(f 1 2 3)` și `(initial-fact)`
(c) se va produce o singură activare și în baza de fapte vom avea doar `(f 1 2 3)`
(d) se va produce o singură activare și în baza de fapte vom avea doar `(initial-fact)`

19. În care limbaj expresia corespunzătoare **nu** va produce eroare?

(a) Haskell: `let x = 10 `div` 0`
(b) CLIPS: `(assert (X (/ 10 0)))`
(c) Prolog: `X is 10 / 0.`
(d) Scheme: `(define X (10 / 0))`

20. Fie următoarea funcție în Scheme, respectiv Haskell:

```
(define (f x) (car (cons x (map f '(1 2 3)))))
f x = head (x : (map f [1, 2, 3]))
```

Alegeți afirmația adevarată:

(a) Apelul `(f 1)` va cicla în Scheme, iar în Haskell va întoarce `1`.
(b) Apelul `(f 1)` va cicla atât în Scheme, cât și în Haskell.
(c) Apelul `(f 1)` va întoarce `1` în Scheme și va cicla în Haskell.
(d) Apelul `(f 1)` va întoarce `1` atât în Scheme, cât și în Haskell.