Test PP Numele ..... 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. Fie următoarea funcție Racket ce generează toate submulțimile de dimensiune `subset-size` ale unei mulțimi set de dimensiune `set-size`:

```
(define (subsets set set-size subset-size current-set result)
  (cond [(zero? subset-size)
         (cons (reverse current-set) result)]
        [(> subset-size set-size) result]
        [else (subsets (cdr set)
                       (sub1 set-size)
                       (sub1 subset-size)
                       (cons (car set) current-set)
                       (subsets (cdr set)
                                (sub1 set-size)
                                subset-size
                                current-set
                                result))]))
```

Ce fel de recursivitate utilizează funcția?

(a) recursivitate pe stivă
(b) recursivitate pe coadă
(c) ambele tipuri de recursivitate
(d) funcția nu este recursivă

2. Ce afișează codul Racket următor?

```
(define (f L)
  (if (or (null? (cdr L)) (null? L))
      1
      (+ 1 (f (cddr L)))))
(f '(1))
(f '(1 2 3 4 5 6 7 8))
```

(a) 1 și eroare
(b) 1 și 4
(c) 1 și 8
(d) 1 și 5

3. La ce se evaluează expresia Racket de mai jos?

```
(map append '((+ 1 1) (+ 2 2)))
```

(a) `((+ 1 1) (+ 2 2))`
(b) `(+ 1 1 + 2 2)`
(c) `(2 4)`
(d) `((2 4))`

4. Care dintre următoarele reprezintă o metodă corectă pentru a verifica în Racket dacă toate elementele listei L sunt adevărate?

(a) `(foldr (lambda (x acc) (and x acc)) #t L)`
(b) `(apply or L)`
(c) `(apply map and L)`
(d) `(null? (filter (lambda (x) x) L))`

5. Care este rezultatul următoarei expresii Haskell?

```
foldl (\y x -> x : y) [] [1, 2, 3, 4, 5]?
```

(a) `[5, 4, 3, 2, 1]`
(b) `[1, 2, 3, 4, 5]`
(c) `[5, [4, [3, [2, [1]]]]]`
(d) eroare

6. Ce afișează codul Racket următor?

```
(define L '(1 2 3))
(define x (cdr L))
(define y (cddr L))
(define z (cdddr L))
(let* ([x y] [y z] [z x])
  (list x y z))
```

(a) `((3) () (3))`
(b) `((3) () (2 3))`
(c) `(2 3 2)`
(d) eroare

7. Care dintre următoarele expresii Haskell produc eroare?

```
1) \x y -> x + 1 / 2 $ 5 2
2) let f = (/ 0) $ 1 in 0
3) f = (+ 2) $ 1
4) \x -> x + 1 / 0
```

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

8. Ce afișează (prin `display` și valoare de retur) codul Racket următor?

```
(define x (delay ((lambda (x) (display x) (+ x x)) 5)))
(define y (delay ((lambda (x) (display x) (+ x x)) 5)))
(define z x)
(* (force x) (force y) (force z))
```

(a) 551000
(b) 5551000
(c) 51000
(d) 5102000

9. În Haskell, care este tipul expresiei de mai jos?

```
let x y = (+ y) in map . x
```

(a) `Num a => a -> [a] -> [a]`
(b) `Num a, Num b => b -> (b -> a) -> [b] -> [a]`
(c) `Num a => a -> (a -> a) -> [a] -> [a]`
(d) `Num a => b -> (b -> a) -> [b] -> [a]`

10. În Haskell, care este tipul expresiei de mai jos?

```
iterate (\x -> (+ (x 2))) (+ 1)
```

(a) `Num a => [a -> a]`
(b) Eroare de sinteză de tip
(c) `Num a => [a]`
(d) `Num a => a -> [a]`

11. Care este tipul următoarei expresii Haskell?

```
g x y z = filter z . y $ x
```

(a) `a1 -> (a1 -> [a2]) -> (a2 -> Bool) -> [a2]`
(b) `a1 -> (a1 -> [a2]) -> (a2 -> Bool) -> [a1]`
(c) `a1 -> (a1 -> [a2]) -> (a1 -> Bool) -> [a1]`
(d) `a2 -> (a1 -> [a2]) -> (a1 -> Bool) -> [a2]`

12. Care este tipul expresiei `T . field`, conform definiției Haskell de mai jos?

```
data T a = T { field :: a }
```

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

13. La ce se va evalua expresia

```
sort [T 2, T 4, T 5, T 6, T 8, T 9]
```

pentru codul Haskell de mai jos?

```
data T = T Int deriving (Eq, Show)
f (T x) = sum [ 1 | a <- [1 .. x], mod x a < 1 ]
instance Ord T where
    x <= y = f x <= f y
```

(a) `[T 2, T 5, T 4, T 9, T 6, T 8]`
(b) `[T 2, T 4, T 5, T 6, T 8, T 9]`
(c) `[T 2, T 4, T 8, T 6, T 9, T 5]`
(d) `[T 2, T 4, T 5, T 8, T 9, T 6]`

14. Care dintre următoarele interogări Prolog generează erori?

```
1) L1 = [pikachu, ratatta], L2 = [charmander, charizard],
   findall((X + Y), (member(X, L1), member(Y, L2)), L).
2) L1 = [pikachu, ratatta], L2 = [charmander, charizard],
   findall(Z, (member(X, L1), member(Y, L2), Z = X + Y), L).
3) L1 = [pikachu, ratatta], L2 = [charmander, charizard],
   findall(Z, (member(X, L1), member(Y, L2), Z is X + Y), L).
```

(a) doar 3)
(b) doar 2)
(c) doar 1)
(d) toate trei

15. Câte variante de satisfacere a scopului `a(X, Y, Z)` există pentru programul de mai jos?

```
p(1). p(2). p(3).
q(3). q(4).
r(1). r(2). r(3).
a(X, Y, Z) :- p(X), !, \+ q(X), q(Y), !, r(Z).
```

(a) 3
(b) 0
(c) 6
(d) 12

16. Ce va afișa interogarea `r(X, Y)` în Prolog pentru programul de mai jos?

```
p(1). p(2).
q(X, Y) :- p(X), p(Y), X < Y.
r(X, Y) :- p(X), \+ q(X, Y).
```

(a) `X = 2.`
(b) `X = Y, Y = 1 ; X = 2, Y = 1 ; X = Y, Y = 2.`
(c) `X = 2, Y = 1.`
(d) `false.`

17. Fie următoarele definiții:

```
fibonacci(0,0).
fibonacci(1,1).
fibonacci(N,F) :-
    N > 1,
    N1 is N - 1,
    N2 is N - 2,
    fibonacci(N1,F1),
    fibonacci(N2,F2),
    F is F1 + F2.
```

Ce vor afișa următoarele interogări?

```
?- fibonacci(6, F).
?- fibonacci(N, 8).
```

(a) Prima interogare va lega `F` la valoarea `8` (`F = 8`), iar a doua va produce eroare (`Arguments are not sufficiently instantiated`).
(b) Ambele interogări vor produce eroare (`Arguments are not sufficiently instantiated`).
(c) Prima interogare va produce eroare, iar a doua va lega `N` la `6`.
(d) Prima interogare va lega `F` la `8`, iar a doua va lega `N` la `6`.

18. Fie următoarea interogare în Prolog:

```
?- List1 = [1, 2, 3, 4, 5, 6], List2 = [2, 4, 8, 10, 1],
|      findall((X/Y), (member(X, List1), member(Y, List2),
|          \+ member(X, List2), \+ member(Y, List1)), L),
|      length(L, N).
```

La ce sunt legate variabilele `L` și `N`?

(a) `L = [3/8, 3/10, 5/8, 5/10, 6/8, 6/10], N = 6.`
(b) `L = [0.375, 0.3, 0.625, 0.5, 0.75, 0.6], N = 6.`
(c) `L = [], N = 0.`
(d) Interogarea generează eroare.

19. Cu ce secvență/secvențe Haskell poate fi comparată următoarea secvență Prolog?

```
forall(member(X,L), X mod 2 =:= 0).
1) null $ filter even lst
2) length lst == length (filter even lst)
3) length lst == length [x | x <- lst, x `mod` 2 == 0]
```

(a) 2) și 3)
(b) 1), 2) și 3)
(c) doar 2)
(d) doar 3)

20. Ce concept din Haskell se regăsește în Prolog?

(a) *pattern matching*
(b) *list comprehensions*
(c) gărzi
(d) evaluare leneșă