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. Ce fel de recursivitate au funcțiile `f` și `g` în Racket?

```
(define (f p L)
  (and (not (null? L))
       (or (p (car L))
           (f p (cdr L)))))
(define (g p L)
  (or (null? L)
      (p (car L))
      (not (g p (cdr L)))))
```

   (a) `f` pe coadă, `g` pe stivă
   (b) ambele pe stivă
   (c) ambele pe coadă
   (d) `f` pe stivă, `g` pe coadă

2. De câte ori se redimensionează crescător stiva pentru apelul `(f 9)` al funcției Racket de mai jos?

```
(define (f n)
  (cond [(zero? n) 0]
        [(even? n) (+ n (f (- n 1)))]
        [(odd?  n) (f (- n 1))]))
```

   (a) 4
   (b) 9
   (c) 8
   (d) 5

3. Pornind de la definiția Racket de mai jos, ce produce evaluarea expresiei `(process '(1 (2 3) 4 (5 (6)) 7))`?

```
(define (process lst)
  (foldr (lambda (x acc)
           (if (list? x)
               (append x acc)
               (cons x acc)))
         '()
         lst))
```

   (a) `(1 2 3 4 5 (6) 7)`
   (b) `(1 2 3 4 5 6 7)`
   (c) `append: contract violation`
   (d) `(1 (2 3) 4 (5 (6)) 7)`

4. Care dintre următoarele funcții este echivalentă cu funcția Haskell `sum . map (* 2)`, unde `sum` calculează suma elementelor unei liste?

   (a) `(* 2) . sum`
   (b) `(* 2) . map sum`
   (c) `map (* 2) . sum`
   (d) `map sum . (* 2)`

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

```
(map (lambda (x) (map x '(1 2 3) '(4 5 6)))
     (list + -))
```

   (a) `'((5 7 9) (-3 -3 -3))`
   (b) `'((5 7) (9 -3) (-3 -3))`
   (c) `'((5 -3) (7 -3) (9 -3))`
   (d) `'(5 7 9 -3 -3 -3)`

6. Ce produce evaluarea expresiei următoare în Racket?

```
(letrec ([len (length lst)]
         [lst '(1 2 3 4 5)])
  len)
```

   (a) Eroare: deși `lst` este vizibil la inițializarea lui `len`, este folosit înainte de propria inițializare
   (b) `5`
   (c) Eroare: `lst` nu este vizibil la inițializarea lui `len`
   (d) `#f`

7. Care este valoarea expresiei Racket de mai jos?

```
(let* ([a 1] [f (lambda (x) (+ a x))])
  (let ([a 5] [x (+ a a)])
    (f x)))
```

   (a) `3`
   (b) `15`
   (c) `11`
   (d) `7`

8. Ce se afișează, atât prin funcția `display`, cât și în urma evaluării, la rularea codului Racket de mai jos?

```
(define x 5)
(define (f x) (display "x ") (delay (+ x 2)))
(define g (delay (display "y ") (+ x 2)))
(list (force (f 2)) (force g) (force g) (force (f 2)))
```

   (a) `x y x '(4 7 7 4)`
   (b) `x y '(4 7 7 4)`
   (c) `x y y x '(7 7 7 7)`
   (d) `x y '(7 7 7 7)`

9. Fie următoarea funcție in Racket:

```
(define (every-nth s n)
  (let loop ([s s] [m 0])
    (cond
      [(stream-empty? s) s]
      [(= (modulo m n) 0)
       (stream-cons (stream-first s)
                    (loop (stream-rest s) (+ m 1)))]
      [else (loop (stream-rest s) (+ m 1))])))
```

   Care vor fi rezultatele evaluării, pe rând, a următoarelor expresii, unde funcția `stream-take` întoarce un flux cu primele câteva elemente ale altui flux, iar funcția `stream->list` transformă un flux într-o listă?

```
(stream->list (stream-take (every-nth naturals 100) 4))
(take (stream->list (every-nth naturals 100)) 4)
```

   (a) `'(0 100 200 300)`; nu se termină
   (b) nu se termină; `'(0 100 200 300)`
   (c) `'(0 100 200 300)`; `'(0 100 200 300)`
   (d) nu se termină; nu se termină

10. De câte ori va fi aplicată funcția `(^ 2)` în cadrul evaluării expresiei Haskell `head $ filter even $ map (^ 2) [1, 2, 3, 4]`?

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

11. Care dintre următoarele definiții va genera eroare, pornind de la tipul Haskell de mai jos?

```
data MyType = A Float Float | B Bool
```

   (a) `f (A x) (B b) = True`
   (b) `f = (A, B, A 1.0)`
   (c) `f (A _ _) (B _) = False`
   (d) `f = [A (12+4) 5, B True]`

12. Care este tipul sintetizat de Haskell pentru funcția de mai jos?

```
f x y z = x y . z
```

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

13. Care ar fi antetul corect pentru a instanția clasa `Functor` cu constructorul de tip `Either` din Haskell?

```
class Functor f where
  fmap :: (a -> b) -> f a -> f b
data Either c d = Left c | Right d
```

(a) `instance Functor (Either c) where`
(b) `instance Functor (Either c d) where`
(c) `instance Functor Either where`
(d) Nu se poate instanția clasa

14. Cunoscând în Haskell:

```
null :: Foldable t => t a -> Bool
takeWhile :: (a -> Bool) -> [a] -> [a]
```

ce fel de polimorfism are funcția `f`?

```
f x l = null (takeWhile x l)
```

(a) Doar parametric
(b) Doar ad-hoc
(c) Atât parametric, cât și ad-hoc
(d) Funcția nu este polimorfică

15. Ce produce interogarea Prolog de mai jos?

```
length(L, 3), append(_, [_, 1 | _], L).
```

(a) 2 soluții (2 legări distincte pentru `L`)
(b) `true.`
(c) `false.`
(d) 3 soluții (3 legări distincte pentru `L`)

16. Fie un graf orientat specificat în Prolog prin fapte de tip `nod/1` și `arc/2`. Care este cea mai precisă interpretare a faptului că interogarea de mai jos **eșuează**?

```
forall(nod(X), (arc(X, _); arc(_, X))).
```

(a) Graful conține noduri izolate.
(b) Graful conține atât noduri în care nu intră nimic cât și noduri din care nu iese nimic.
(c) Graful nu conține cicluri.
(d) Graful nu conține lanțuri de lungime 2.

17. Fie graful următor specificat în Prolog prin fapte de tip `arc/2`:

```
arc(a, b). arc(a, c). arc(b, d). arc(c, e). arc(c, g).
```

Ce rezultat produce interogarea:

```
findall(X, (arc(P, _), !, arc(P, X)), L).
```

(a) `L = [b, c].`
(b) `L = [c, g].`
(c) `L = [b].`
(d) `L = [b, c, d, e, g].`

18. Care dintre următoarele implementări din Prolog îl leagă pe `X` la `7/5` în urma interogării `f(3/5, 4/5, X).`?

```
(1) f(A/X, B/X, C/X) :- C is A+B.
(2) f(A/B, C/D, E/F) :- B =  D, B =  F, E is A+C.
(3) f(A/B, C/D, E/F) :- B == D, B == F, E is A+C.
(4) f(A/B, C/D, E/F) :- B == D, B =  F, E is A+C.
```

(a) (1), (2) și (4)
(b) Doar (2)
(c) (1) și (3)
(d) (1) și (2)

19. Pornind de la definițiile de mai jos, în care `g` adaugă 1 și `f` înmulțește cu 2, alegeți ordinea rezultatelor intermediare obținute în timpul evaluării aplicațiilor `(r '(1 2 3))`, respectiv `(h [1, 2, 3])`.

```
(define r (compose (curry map f) (curry map g)))  ;; Racket
h = map f . map g                                 -- Haskell
```

(a) 2, 3, 4, 4, 6, 8, respectiv 2, 4, 3, 6, 4, 8
(b) 2, 4, 3, 6, 4, 8, respectiv 2, 3, 4, 4, 6, 8
(c) 2, 3, 4, 4, 6, 8, respectiv 2, 3, 4, 4, 6, 8
(d) 2, 4, 3, 6, 4, 8, respectiv 2, 4, 3, 6, 4, 8

20. Care dintre următoarele definiții sunt valide (în limbajele aferente)?

```
(define (same x x) #t)  ;; Racket
same x x = True         -- Haskell
same(X, X).             %% Prolog
```

(a) Doar cea în Prolog
(b) Cele în Haskell și Prolog
(c) Cele în Racket și Haskell
(d) Doar cea în Haskell