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. Pentru definiția Racket de mai jos, ce adâncime maximă atinge stiva în timpul evaluării expresiei `(f 5)`? Obs: dacă funcția ar fi recursivă pe coadă, toate apelurile lui `f` s-ar afla la adâncime 0, datorită *tail-call optimization*.

```
(define (f x)
  (cond [(< x 0) x]
        [(odd? x) (f (- x 1))]
        [else (+ (f (- x 3)) (f (- x 4)))]))
```

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

2. Alegeți cea mai precisă afirmație despre următoarea funcție recursivă în Racket:

```
(define (f x acc)
  (if (zero? x) acc
      (f (sub1 x) (add1 acc))))
```

   (a) `f` este recursivă pe coadă pentru că nu mai realizează alte operații după aplicația recursivă.
   (b) `f` este recursivă pe coadă pentru că nu mai realizează alte operații după aplicația recursivă și pentru că folosește un acumulator (fără acumulator nu am avea recursivitate pe coadă).
   (c) `f` este recursivă pe coadă pentru că folosește un acumulator.
   (d) `f` este recursivă pe stivă.

3. Care dintre funcționalele `map`, `foldl`, `foldr`, `apply` poate înlocui semnul întrebării în expresia Racket de mai jos astfel încât aceasta să se evalueze la `'((1))`?

```
(? (lambda (x y) (y x)) (list 1) (list list))
```

   (a) `map` și `apply`
   (b) Doar `map`
   (c) Doar `foldl`
   (d) `foldl` și `foldr`

4. Care este valoarea expresiei Racket de mai jos?

```
((foldr compose
        (lambda (x) x)
        (list (curry + 1) (curry * 2)))
 5)
```

   (a) 11
   (b) 12
   (c) 10
   (d) Eroare

5. Ce calculează următoarea expresie în Racket, unde `mat` este o matrice, reprezentată ca listă de linii (i.e. listă de liste de numere)?

```
(apply + (apply append mat))
```

   (a) Suma numerelor din matrice
   (b) Suma fiecărei coloane din matrice
   (c) Suma fiecărei linii din matrice
   (d) Eroare

6. La ce se evaluează fiecare dintre următoarele două expresii în Racket?

```
(define x 3)
(letrec ([f (lambda () (+ x 3))]  ; (1)
         [x 6])
  (f))
(let ([f (delay (+ x 3))]         ; (2)
      [x 6])
  (force f))
```

   (a) Prima se evaluează la 9, iar a doua la 6.
   (b) Prima generează eroare, iar a doua se evaluează la 6.
   (c) Ambele se evaluează la 6.
   (d) Ambele se evaluează la 9.

7. Care dintre următoarele expresii din Racket este echivalentă cu:

```
(let* ([x 10]
       [y (lambda () (+ x 1))])
  (y))
```

   (a)
```
((lambda (x)
   ((lambda (y) (y)) (lambda () (+ x 1))))
 10)
```
   (b)
```
((lambda (x)
   ((lambda (y) (lambda () (+ x 1))) (y)))
 10)
```
   (c)
```
((lambda (x y) (y))
 10 (lambda () (+ x 1)))
```
   (d)
```
((lambda (x y) 10 (lambda () (+ x 1)))
 (y))
```

8. De câte ori se calculează corpul funcției `calcul` la evaluarea expresiei Racket de mai jos?

```
(stream-first (stream-cons (calcul 0)
                           (stream-cons (calcul 1)
                                        empty-stream)))
```

   (a) O singură dată
   (b) De 2 ori
   (c) De 3 ori
   (d) Niciodată

9. Pentru programul Haskell de mai jos, de câte ori se calculează corpul funcției `f` la evaluarea expresiei `length [a 2, a 2, a 2, b, b]`?

```
f x y = x + y
a = f 1
b = f 2 3
length [] = 0
length (_:xs) = 1 + length xs
```

   (a) Niciodată
   (b) De 2 ori
   (c) De 4 ori
   (d) De 5 ori

10. Fie următoarele definiții în Haskell:

```
xs = map (+ 1) [1 .. 10]
f (8:9:_) = True
```

Câte elemente ale listei `xs` vor fi calculate dacă se evaluează expresia `(f xs)`?

   (a) 1
   (b) 2
   (c) 10
   (d) Niciunul

11. Pentru programul Haskell de mai jos, ce tip va avea expresia `[x, y]`?

```
data MyData a b
    = A a (MyData a b)
    | B b (MyData a b)
    | C a b
x = A id
y = B "id"
```

   (a) `[MyData (a -> a) [Char] -> MyData (a -> a) [Char]]`
   (b) `Char b => [MyData a [b] -> MyData a [b]]`
   (c) Eroare de tip
   (d) `[(a -> a) -> [Char] -> MyData (a -> a) [Char]]`

12. Care este tipul listei Haskell `[foldl, foldr]`?

    (a) `Foldable t => [(b -> b -> b) -> b -> t b -> b]`
    (b) `Foldable t => [(b -> a -> b) -> b -> t a -> b]`
    (c) `Foldable t => [(a -> b -> b) -> b -> t a -> b]`
    (d) Eroare de tip

13. Funcția `length` din Haskell are tipul

    `Foldable t => t a -> Int`

    Acest lucru înseamnă că:

    (a) `length` este polimorfică ad-hoc în variabila de tip `t` și polimorfică parametric în `a`.
    (b) `length` este polimorfică parametric în variabila de tip `t` și polimorfică ad-hoc în `a`.
    (c) `length` este polimorfică parametric exclusiv.
    (d) `length` este polimorfică ad-hoc exclusiv.

14. Fie tipul de date definit astfel în Haskell:

    `type Func a = a -> a`

    Dacă am încerca să instanțiem clasa `Functor` cu constructorul `Func`, tipul lui `fmap` ar fi:

    `(a -> b) -> Func a -> Func b` sau
    `(a -> b) -> (a -> a) -> (b -> b)`

    În câte moduri **diferite** poate calcula `fmap` funcția cu tipul `(b -> b)`, exceptând `undefined`?

    (a) Într-un singur mod, întorcând funcția identitate, deoarece nu putem folosi primele două funcții pentru a o construi pe a treia.
    (b) În două moduri, întorcând fie funcția identitate, fie cea rezultată prin compunerea primelor două funcții.
    (c) O infinitate.
    (d) Niciunul.

15. Se dă următorul program Prolog:

```
a(1). a(2). a(3). a(4).
b(1). b(2). b(3). b(4).
p1(X, Y) :- a(X), !, b(Y).
p2(X, Y) :- !, a(X), b(Y).
p3(X, Y) :- a(X), b(Y), !.
p4(X, Y) :- a(X), !; b(Y).
```

    Alegeți varianta corectă:

    (a) Interogarea `p2(X, Y)` va lega atât pe `X`, cât și pe `Y`, la toate valorile posibile.
    (b) Interogările `p3(X, Y)` și `p4(X, Y)` vor avea un comportament identic.
    (c) Interogarea `p4(X, Y)` va lega pe `Y` la toate valorile posibile.
    (d) Interogările `p1(X, Y)` și `p4(X, Y)` vor lega pe `X` la valoarea `1`, apoi vor eșua (nu vor lega pe `Y` la nicio valoare).

16. De câte ori este satisfăcută interogarea `pred(R)`, conform regulii de mai jos?

```
pred(R) :- append(A, B, [1,X,Y]), member(R, A), member(R, B).
```

    (a) De 4 ori
    (b) Niciodată
    (c) De 2 ori
    (d) De 3 ori

17. Ce va afișa Prolog pentru interogarea de mai jos?

```
A=[1,4], B=[1,2,3], forall((member(X, A), !), member(X, B)).
```

    (a) `A = [1, 4], B = [1, 2, 3].`
    (b) `false.`
    (c) `X = 1.`
    (d) `X = 1; X = 4.`

18. Care este rezultatul următoarei interogări în Prolog:

```
L = [2 + 3, 6 * 5, 8 / 4, 12 - 9, 18 / 3, 21 - 7],
    findall(Y, (member(X, L), X mod 3 =:= 0, Y is X), Res).
```

    (a) `L = [2+3, 6*5, 8/4, 12-9, 18/3, 21-7],`
        `Res = [30, 3, 6].`
    (b) `L = [5, 30, 2, 3, 6, 14],`
        `Res = [30, 3, 6].`
    (c) `L = [2+3, 6*5, 8/4, 12-9, 18/3, 21-7],`
        `Res = [6*5, 12-9, 18/3].`
    (d) `L = [5, 30, 2, 3, 6, 14],`
        `Res = [5, 2, 14].`

19. În care dintre limbajele studiate putem scrie cod care utilizează **recursivitate pe coadă**?

    (a) Racket, Haskell și Prolog
    (b) Racket și Haskell
    (c) Racket și Prolog
    (d) Doar Racket

20. În care dintre limbajele studiate am discutat despre conceptul de **unificare**?

    (a) Unificarea expresiilor de tip în Haskell și a expresiilor în Prolog.
    (b) Unificarea listelor imbricate în Racket și a expresiilor de tip în Haskell.
    (c) Unificarea listelor imbricate în Racket și a expresiilor în Prolog.
    (d) În niciun limbaj.