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 afișează următorul cod Racket:

```
(define (abc L)
  (if (null? L) L
      (cons (cond [(eq? (car L) 'a) 'b]
                  [(eq? (car L) 'b) 'c]
                  [(eq? (car L) 'c) 'b]
                  [else (car L)])
            (abc (cdr L)))))
(abc (abc '(a b c c b a d d b b c a)))
```

   (a) `(c b c c b c d d b b c c)`
   (b) `(b c b b c b d d c c b b)`
   (c) `(c b c b b c d d b b c c)`
   (d) `(c b b b b c d d b b b c)`

2. Ce complexitate temporală, respectiv spațială (spațiu suplimentar pe stivă), are următoarea implementare Racket, unde `x` si `p` sunt două numere naturale?

```
(define (f x p)
  (if (= p 0)
      1
      (let* ([a (f x (quotient p 2))]
             [aa (* a a)])
        (if (= (modulo p 2) 1)
            (* aa x)
            aa))))
```

   (a) $O(\log p)$ și $O(\log p)$
   (b) $O(p)$ și $O(1)$
   (c) $O(\log p)$ și $O(1)$
   (d) $O(p^2)$ și $O(p)$

3. Se dă următoarea funcție în Racket:

```
(define (f L)
  (foldr (lambda (x y)
           (+ (car x) y))
         0
         L))
```

   Care ar putea fi o definiție echivalentă pentru aceasta?

   (a) `(define (f L) (apply + (map car L)))`
   (b) `(define (f L) (map (lambda (x y) (+ (car x) y)) L))`
   (c) `(define (f L) (foldr + 0 L))`
   (d) `(define (f L) (foldl + 0 L))`

4. Considerând că parametrul funcției `sumPrimes` are tipul `[[Int]]`, care dintre următoarele definiții calculează suma tuturor numerelor prime din acesta? Vom considera că există funcția `isPrime`, având următorul tip: `isPrime :: Int -> Bool`.

   (a) `sumPrimes = sum . map sum . map (filter isPrime)`
   (b) `sumPrimes = sum . filter isPrime`
   (c) `sumPrimes = sum . map sum . filter isPrime`
   (d) `sumPrimes = map (sum . filter isPrime)`

5. Care este funcționala adecvată pentru rescrierea codului următor?

```
(define (foo L)
  (if (null? L) a
      (b (car L) (foo (cdr L)))))
```

   (a) `foldr`
   (b) `map`
   (c) `foldl`
   (d) `filter`

6. La ce se evaluează expresia Haskell următoare?

```
last . zip "cat" $ "fish"
```

   (a) `('t', 's')`
   (b) `[('t', 'h')]`
   (c) `("cat", "fish")`
   (d) `'h'`

7. Care va fi rezultatul rulării următorului cod Racket:

```
(define a ( (lambda (x)
              (lambda (y)
                (+ x 1)))  5))
(a 'b)
```

   (a) `6`
   (b) `5`
   (c) o promisiune
   (d) eroare: `x` nu este vizibil la aplicarea lui `a`

8. Fie definiția următoarei funcții în Racket:

```
(define (f x) (lambda () (+ x 3)))
```

   Care este diferența dintre aplicațiile `(f 5)` și `((f 5))`?

   (a) Prima se evaluează la o funcţie, iar a doua, la `8`.
   (b) Prima se evaluează la `8`, iar a doua produce eroare.
   (c) Prima produce eroare, iar a doua se evaluează `8`.
   (d) Ambele se evaluează la `8`.

9. Dacă dorim să implementăm în Racket funcția de compunere a altor două funcții, care implementare este corectă?

   (a) `(define (comp f g) (lambda (x) (f (g x))))`
   (b) `(define (comp f g) (let ((lambda (x) g)) (f x)))`
   (c) `(define (comp f g) (f (g comp)))`
   (d) `(define (comp f g) (f g))`

10. Care este tipul expresiilor:

```
:t (+ 1) . (0 <)
:t (0 <) . (+ 1)
```

   (a) eroare și `(Num a, Ord a) => a -> Bool`
   (b) `Num c => c -> c` și eroare
   (c) `(Num a, Ord a) => a -> Bool` și `(Num a, Ord a) => a -> Bool`
   (d) `Num c => c -> c` și `(Num a, Ord a) => a -> Bool`

11. Care este tipul următoarei funcții Haskell?

```
f x y z = map x . y $ z
```

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

12. Care este tipul expresiei Haskell `(. map odd)`?

   (a) `Integral a => ([Bool] -> b) -> [a] -> b`
   (b) `Integral a => ([a] -> [b]) -> [a] -> [b]`
   (c) `Integral a => [a] -> ([Bool] -> b) -> b`
   (d) `Integral a => [a] -> (Bool -> b) -> [b]`

13. Câte funcții de tip `a -> a -> a` (fără alte constrângeri asupra tipurilor) există în sens matematic (ca mulțimi de asocieri intrare-ieșire)?

   (a) 3
   (b) o infinitate
   (c) 1
   (d) 0

14. Care este semnificația codului următor?

```
class Num a => Fractional a where
```

(a) Pentru ca tipul `a` să fie membru al clasei `Fractional`, el trebuie să fie membru al clasei `Num`.
(b) Dacă tipul `a` este membru al clasei `Num`, atunci el este și membru al clasei `Fractional`.
(c) `Fractional a` este un tip de date care aparține clasei `Num a`.
(d) Clasa `Num a` poate fi instanțiată numai de tipuri de numere fracționare.

15. Fie interogarea:

```
?- L1 = [1, 2, 3], L2 = [a, b], K = 3,
|     findall((K, V), (member(K, L1), member(V, L2)), O).
```

La ce este legată variabila `O`?

(a) `O = [(3, a), (3, b)]`.
(b) Interogarea dă eroare.
(c) Interogarea întoarce `false`.
(d) `O = [(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)]`.

16. Fie următoarele definiții:

```
a(1). b(1). c(2).
p(X) :- a(X), b(X), !.
p(X) :- c(X).
```

Ce va afișa următoarea interogare?

```
?- p(X).
```

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

17. Fie următoarele definiții:

```
man(barber).
man(john).
shaves(barber, X) :- man(X), \+ shaves(X, X).
```

Ce va afișa următoarea interogare?

```
?- shaves(barber, X).
```

(a) `ERROR: Out of local stack`
(b) `X = barber.`
(c) `X = barber; X = john.`
(d) `false.`

18. Care dintre următoarele două interogări vor fi satisfăcute (vor întoarce `true`)?

```
?- X == Y, X = Y.
?- X = Y, X == Y.
```

(a) A doua.
(b) Ambele.
(c) Niciuna.
(d) Prima.

19. Câte rezultate va avea următoarea interogare? (de câte ori va fi satisfăcută?)

```
?- L1 = [1,2,3,4,5], L2 = [1,3,5],
|     \+ (member(X, L2), \+ member(X, L1)).
```

(a) O dată.
(b) Niciodată.
(c) De două ori.
(d) De trei ori.

20. Fie următoarele definiții în cadrul unui program Prolog:

```
copil(andrei, ion).
copil(ion, ana).
%
descendent1(X, Y) :- copil(X, Y).
descendent1(X, Y) :- copil(X, Z), descendent1(Z, Y).
%
descendent2(X, Y) :- descendent2(Z, Y), copil(X, Z).
descendent2(X, Y) :- copil(X, Y).
```

Care sunt rezultatele interogărilor următoare:

```
?- descendent1(andrei, ana).
?- descendent2(andrei, ana).
```

(a) Prima interogare va întoarce `true`, iar a doua va cicla.
(b) Prima interogare va cicla, iar a doua va întoarce `true`.
(c) Ambele interogări vor cicla.
(d) Ambele interogări vor întoarce `true`.