Funkcionális programozás elméleti teszt

MINTA

FIGYELEM: *A jelenlegi feladatsor egy minta és csupán 7 kérdésből áll. A feladatsor azt hivatott bemutatni, hogy milyen jellegű kérdések várhatóak a vizsga elméleti kvízében. A vizsgán 12 kérdésből álló feladatsor kerül kiadásra.*

Olvashatóan, nyomtatott nagybetűvel töltsük ki:

Név: _____

NEPTUN: _____

Válaszd ki a helyes választ a megadottak közül! Minden kérdésnél egyetlen helyes válasz létezik, **javított megoldás nem fogadható el**. Legalább **7 helyes** válasz elérése kötelező az elméleti teszt teljesítéséhez. A jelölés egyértelmű legyen (karikázás vagy X), különben nem tudjuk elfogadni!

Kérdések (részlet)

1. Melyik kifejezéssel ekvivalens a `[g x | x <- xs, f x]` listakifejezés?

   a) `(filter g . map f) xs`
   b) `(filter f . map g) xs`
   c) `(map f . filter g) xs`
   d) `(map g . filter f) xs`

2. Melyik kifejezés típusozható az alábbiak közül?

   a) `length [] ++ [1]`
   b) `length "valami" / 2`
   c) `take 1`
   d) A felsoroltak egyike sem.

3. Mi a `foldr` függvény típusa?

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

4. Az alábbi definíciók közül melyik ad meg totális függvényt?

   a) `f (x,_)  = x`
   b) `f (x:_)  = x`
   c) `f (x:[]) = x`
   d) `f (x,[]) = x`

5. Mi **nem** lehet a típusa a `Saturday` adatkonstruktornak az alábbiak közül?

   ```
   data SomeDays a b = Monday a | Saturday a b
   ```

   a) `Int -> Char -> SomeDays Int Char`
   b) `String -> Int -> SomeDays Int String`
   c) `Char -> Char -> SomeDays Char Char`
   d) `Int -> Bool -> SomeDays Int Bool`

6. Melyik kifejezésre illeszkedik az `(x,y):z` minta?

   a) `[(even, odd)]`
   b) `(['a'], ['b'])`
   c) `[[1,2], []]`
   d) `(1, 2, [3])`

7. Az alábbi függvények közül melyiknek lehet **totális** (nem parciális) definíciót adni (undefined, error és végtelen rekurzió használata nélkül)?

   a) `f :: [a] -> a`
   b) `g :: Maybe a -> a`
   c) `h :: (a,a) -> a`
   d) `i :: (a -> a) -> a`