Haskell 3.

Egyéni típusok

- **Típusszinonimák** – névvel ellátott összetett típust definiál már meglévő típusokból, nem újat hoz létre!

  type Pair = (Int, Int)

  osszead :: Pair -> Pair -> Pair

  osszead (a,b) (c,d) = (a+c, b+d)

- **Valódi (felsorolás-) típusok**
  - A data kulcsszóval deklarálhatjuk őket
  - pl. a Haskell beépített *Bool* típusának definíciója

    data Bool = False | True
  - **|**-nal elválasztva adhatjuk meg a felvehető értékeket
  - pl.: ***data Color = Fekete | Feher | Lila | Zold | Sarga***

1. Írjunk egy *Hetnapja* típust, amely a hét napjait sorolja fel! Majd egy függvényt, amely paraméterként a hét egy napját kapja, és eldönti hétvége-e! **(hetnapja.hs)**

```
data Hetnapja = Hetfo | Kedd | Szerda |
Csutortok | Pentek | Szombat | Vasarnap


hetvege :: Hetnapja -> Bool
hetvege Szombat = True
hetvege Vasarnap = True
hetvege _ = False
```

- **Polimorf típusok**

  pl.:

  polisize :: [a] -> Int

  polisize [] = 0

  polisize (h:t) = 1 + polisize t

2. Definiálj egy *Sikidom* típust, ami egy négyzet, egy téglalap vagy egy kör szükséges (valós típusú) adatait tárolja. Írj két függvényt amik a paraméterként kapott síkidomok területét illetve kerületét számítják ki! **(hs_9.hs)**

type Oldal = Float

type Sugar = Float

data Sikidom = Teglalap Oldal Oldal | Negyzet Oldal | Kor Sugar

```
terulet::Sikidom -> Float
terulet (Teglalap a b) = a*b
terulet (Negyzet a) = a*a
terulet (Kor a) = a*a*3.14
```

```
kerulet::Sikidom>Float
kerulet (Teglalap a b) = 2*(a+b)
kerulet (Negyzet a) = 4*a
kerulet (Kor a) = 2*a*3.14
```

10. Definiálj egy bináris fa típust, ami a belső pontjaiban és a leveleiben is egészek tárolására képes. Írj egy függvényt, ami bejárja a fát és kigyűjti a benne tárolt elemeket egy listába. **(hs_10.hs)**

```
data BinTree a = NIL | Leaf a | Branch a
(BinTree a) (BinTree a)


preorder::BinTree a -> [a]

preorder NIL = []

preorder (Leaf x) = [x]

preorder (Branch x l r) = [x] ++ preorder l ++
preorder r
```

```
inorder::BinTree a -> [a]
inorder NIL = []
inorder (Leaf x) = [x]
inorder (Branch x l r) = inorder l ++ [x] ++
inorder r
```

```
depth::BinTree a -> Int
depth NIL = 0
depth (Leaf x) = 1
depth (Branch x l r) = max (depth l) (depth r) + 1

printTree::BinTree [Char] -> IO()
printTree (Leaf x) = putStrLn x
printTree (Branch x l r) = putStrLn x
```

-- 1. - Polimorf binfa bejárása –---(fabejaras.hs)------

```
data BinFa a = Level a | Belsopont (BinFa a) (BinFa a)


{-    Input:
    Belsopont (Level 2) (Belsopont (Level 3) (Level 4))
  -}


bejar:: (BinFa a) -> [a]
bejar (Level x) = [x]
bejar (Belsopont bal jobb) = bejar bal ++ bejar jobb
```

-- 2. - Egyszerű fa prefix bejárása ----------

```
data Tree = Nil | Node Int Tree Tree


{-   Input:
    Node 4 (Node (-3) (Node 1 Nil Nil) Nil) (Node 2 Nil Nil)  -}


bejarT:: Tree -> [Int]
bejarT Nil = []
bejarT (Node x bal jobb) = [x] ++ (bejarT bal) ++ (bejarT jobb)
```