Demonstrature 1

Definicija 1. Neka su D i K neprazni skupovi. Postupak f koji svakom elementu skupa D pridružuje točno jedan element skupa K zovemo funkcija sa D u K i pišemo

$$f : D \to K \qquad ili \qquad x \mapsto f(x),\ x \in D.$$

Skup D zovemo domena, a skup K kodomena funkcije f.

Primjer 1. Navedimo nekoliko primjera funkcija:

a) $f : \mathbb{N} \to \mathbb{N},\ f(x) = x + 1$

b) $g : \mathbb{N} \to \mathbb{Z},\ g(x) = x - 2$

c) $h : \mathbb{R} \to \mathbb{R},\ h(t) = t^2$

d) $s : \mathbb{R} \to \{7\},\ s(x) = 7$

Primjer 2. Pogledajmo kako definirati gore navedene funkcije u Haskellu:

```
-- Int predstavlja skup Z
f :: Int -> Int
f x = x + 1

g :: Int -> Int
g x = x - 2

-- može i Double umjesto Float
h :: Float -> Float
h t = t^2

s :: Float -> Int
s x = 7
```

Napomena 1. Primijetimo da smo i za skup $\mathbb{N}$ i za skup $\mathbb{Z}$ koristili oznaku `Int`. Razlog tome je što Haskell ne razlikuje ova dva skupa. Pravilo iz matematičke definicije funkcije koje kaže da vrijednost funkcije mora biti moguće izračunati za svaki element domene ovdje možemo "zaobići" tako da jednostavno pretpostavimo da funkciji nikada nećemo slati argument koji je izvan njene domene.

Napomena 2. Ukoliko radimo s funkcijama koje za domenu imaju skup $\mathbb{N}$, često ćemo morati definirati što se događa ako funkciji pošaljemo broj 0 (više o ovome kasnije).

Napomena 3. Skup $\mathbb{R}$ u Haskellu možemo reprezentirati na dva načina:

- `Float` označava decimalne brojeve jednostruke preciznosti
- `Double` označava decimalne brojeve dvostruke preciznosti

Na ovom kolegiju će nam jednostruka preciznost biti sasvim dovoljna. Ukoliko u zadatku nije naglašeno da je preciznost jako važna, koristit ćemo `Float`.

Napomena 4. Osim što može raditi s brojevima, Haskell može raditi i s drugim objektima koji nemaju standardiziranu reprezentaciju u matematici. Jedan primjer takvog objekta je *slovo* - preciznije, bilo koji znak koji možemo unijeti tipkovnicom. Svi takvi znakovi nalaze se u skupu `Char`. Primijetimo da se u Haskellu svi dosad spomenuti tipovi podataka pišu velikim početnim slovom.

Napomena 5. Ponekad nam nije bitno kojeg tipa je argument funkcije te onda koristimo tzv. opće tipove podataka. Ovakvi tipovi podataka se uvijek pišu malim početnim slovom - najčešće `a`, `b` ili `t`. Unutar potpisa funkcije (retka u kojem navodimo domenu i kodomenu) možemo naglasiti da se radi o brojčanom tipu podatka (`Num`), tipu podatka s definiranim uređajem (`Ord`), itd. Više o ovome možete vidjeti u prezentaciji s drugog predavanja.

Primjer 3. Prisjetimo se funkcije s iz Primjera 1. Njena domena je cijeli skup $\mathbb{R}$, dok joj je kodomena skup koji sadrži broj 7. Pretpostavimo da nam nije eksplicitno rečeno da funkcija prima realan broj kao argument, nego joj argument može biti bilo što. Tada ga možemo jednostavno navesti kao objekt nekog općeg tipa:

```
s :: a -> Int
s x = 7
```

Nadalje, budući da nigdje unutar tijela funkcije ne pozivamo njen argument, možemo umjesto njega staviti znak _ (*underscore*).

```
s :: a -> Int
s _ = 7
```

Ovo svojstvo će nam biti vrlo korisno kada budemo radili s funkcijama više varijabli te sa složenijim tipovima podataka kao što su binarna stabla.

Primjer 4. Kao što znamo, u matematici postoje konstante - važni brojevi koji imaju dogovorenu oznaku. Neke od njih uključuju broj e, broj π te imaginarnu jedinicu i. U Haskellu ne postoje konstante u klasičnom smislu, no možemo ih definirati kao konstantne funkcije koje nemaju domenu:

```
sedam :: Int
sedam = 7

pi :: Float
pi = 3.14
```

Primjer 5. Budući da je Haskell vrlo osjetljiv kada su u pitanju tipovi podataka, važno je pretvoriti argument funkcije iz njegovog trenutnog tipa u onaj tip koji nam je potreban. Ovo često može prouzrokovati grešku kada funkciji šaljemo argument tipa `Int`, a unutar tijela funkcije pretpostavljamo da je argument tipa `Float`:

```
f :: Int -> Float
f x = x + 4

g :: Float -> Float
g x = x + 4
```

Funkcija f će prouzrokovati grešku prije nego što ju uopće pozovemo jer vraća zbroj dvaju cijelih brojeva, a Haskell očekuje da će ona vratiti racionalan broj. S druge strane, funkcija g će raditi čak i ako ju pozovemo s cjelobrojnim argumentom - ako nismo eksplicitno rekli da se radi o argumentu tipa `Int`.

```
print $ g 5.0  -- ovo radi
print $ g 5    -- ovo radi

-- ovo ne radi
let sedam = 7 :: Int
print $ g sedam
```

Primijetimo da funkcija g zbraja argument tipa `Float` s brojem 4, koji je očito tipa `Int`, pa bi ova funkcija trebala javljati grešku. Međutim, Haskell može po potrebi pretvoriti cijele brojeve u tip `Float`, ali samo ako nismo eksplicitno naveli da su tipa `Int`.

Nadalje, pretpostavimo da ne smijemo mijenjati potpis funkcije f, ali nam iz nekog razloga treba rezultat koji ona vraća. Tada ćemo pretvoriti njen argument u `Float` koristeći funkciju `fromIntegral`:

```
f :: Int -> Float
f x = fromIntegral x + 4
```

U slučaju da trebamo napraviti obrnuto, odnosno pretvoriti argument tipa `Float` u `Int`, koristit ćemo funkcije `round`, `floor` ili `ceiling`.