7.8. Aurredefinitutako funtzio batzuk zerrendentzat

Aurreko ataletan, errekurtsibitatea eta murgilketa erabiliz funtzio asko definitu ditugu. Funtzio horietako batzuk dagoeneko Haskell lengoaian definituta daude (izen desberdin batekin). Dagoeneko Haskell lengoaian definituta dauden funtzioei aurredefinitutako funtzioak deitzen zaie. Atal honetan, zerrendentzat Haskell-en aurredefinituta dauden funtzio batzuk aipatuko ditugu. Gauza bera egiten duten funtzioak defini ditzakegun arren, ondo dator aurredefinitutako funtzioak ezagutzea, horrela ez baitaukagu geuk definitutako funtzioak dituen fitxategia eskura eduki beharrik.

Aipatuko diren funtzio batzuk oinarrizkoa den Prelude moduluan daude definituta. Beste funtzio batzuk, aldiz, Data.List moduluan daude definituta eta funtzio horiek erabiltzeko modulu hori inportatu beharko da.

7.8.1. Prelude oinarrizko modulukoak diren funtzioak

Atal honetan aipatuko diren funtzioak oinarrizkoa den Prelude moduluan daude definituta eta, ondorioz, beti erabilgarri daude.

head :: [t] -> t
Zerrendako lehenengo elementua itzuliko du.

head [4, 6, 8] → 4 head [] → error

tail :: [t] -> [t]
Zerrendako lehenengo elementua kenduz geratzen den zerrenda itzuliko du.

tail [4, 6, 8] → [6, 8] tail [] → error

last :: [t] -> t
Zerrendako azkeneko elementua itzuliko du.

last [4, 6, 8] → 8 last [] → error

init :: [t] -> [t]
Zerrendako azkeneko elementua kenduz geratzen den zerrenda itzuliko du.

init [4, 6, 8] → [4, 6] init [] → error

null :: [t] -> Bool
Zerrenda hutsa al den erabakiko du.

null [4, 6, 8] → False null [] → True

length :: [t] -> Int
Zerrendaren luzera (elementu kopurua) itzuliko du.

length [4, 6, 8] → 3 length [] → 0

sum :: Num t => [t] -> t
Zerrendako elementuen batura itzuliko du. Elementuek zenbakizkoak izan beharko dute.

sum [4, 6, 8] → 18 sum [] → 0

product :: Num t => [t] -> t
Zerrendako elementuen biderkadura itzuliko du. Elementuek zenbakizkoak izan beharko dute.

product [4, 6, 10] → 240 product [] → 1

reverse :: [t] -> [t]
Alderantzizko zerrenda itzuliko du.

reverse [4, 6, 8] → [8, 6, 4] reverse [] → []

++ :: [t] -> [t] -> [t]
Bi zerrenda emanda, bi zerrendak elkartzen edo kateatzen ditu zerrenda bakarra osatuz eta ordena mantenduz.

```
[5, 2] ++ [4, 9, 8] → [5, 2, 4, 9, 8]
[] ++ [4, 9, 9, 8] → [4, 9, 9, 8]
[5, 2] ++ [] → [5, 2]
```

Prefixu eran ere erabil daiteke:

```
(++) [5, 2] [4, 9, 8] → [5, 2, 4, 9, 8]
(++) [] [4, 9, 9, 8] → [4, 9, 9, 8]
(++) [5, 2] [] → [5, 2]
```

`elem` :: Eq t => t -> [t] -> Bool
Elementu bat zerrenda batean al dagoen erabakiko du. Elementuentzat berdina izatea zer den definituta egon beharko du.

```
9 `elem` [4, 9, 9, 8] → True
7 `elem` [4, 9, 9, 8] → False
9 `elem` [] → False
```

Prefixu eran ere erabil daiteke:

```
(elem) 9 [4, 9, 9, 8] → True
(elem) 7 [4, 9, 9, 8] → False
(elem) 9 [] → False
```

`notElem` :: Eq t => t -> [t] -> Bool
Elementu bat zerrenda batean ez al dagoen erabakiko du. Elementuentzat berdina izatea zer den definituta egon beharko du.

```
9 `notElem` [4, 9, 9, 8] → False
7 `notElem` [4, 9, 9, 8] → True
```



```
9 `notElem` [] → True

9 `notElem` [4, 9, 9, 8] eta
not (9 `elem` [4, 9, 9, 8]) baliokideak dira
```

Prefixu eran ere erabil daiteke:

```
(notElem) 9 [4, 9, 9, 8] → False
(notElem) 7 [4, 9, 9, 8] → True
(notElem) 9 [] → True
```

take :: Int -> [t] -> [t]
Zenbaki bat eta zerrenda bat emanda, zerrendaren hasieratik zenbakiak adierazten duen adina elementu hartuz osatutako zerrenda itzuliko du. Zenbakia zerrendaren luzera baino handiagoa bada, zerrenda osoa itzuliko da. Zenbakia zero edo negatiboa baldin bada, zerrenda hutsa itzuliko da. Zenbakiak Int motakoa izan behar du.

```
take 2 [4, 6, 8] → [4, 6]
take 5 [4, 6, 8] → [4, 6, 8]
take (-2) [4, 6, 8] → []
take 0 [4, 6, 8] → []
```

takeWhile :: (t -> Bool) -> [t] -> [t]
Balio Boolearra itzultzen duen funtzio bat eta zerrenda bat emanda, funtzioak False balioa ematen dion zerrendako lehenengo elementura arteko elementuez osatutako zerrenda itzuliko du. Beraz, funtzioak False ematen dion elementu bat aurkitu arte edo zerrenda bukatu arte jarraituko du.

```
takeWhile (< 3) [1, 2, 3, 4, 1, 2, 3, 4] == [1, 2]
takeWhile (< 9) [1, 2, 3] == [1, 2, 3]
takeWhile (< 0) [1, 2, 3] == []
```

drop :: Int -> [t] -> [t]
Zenbaki bat eta zerrenda bat emanda, zerrendaren hasieratik zenbakiak adierazten duen adina elementu kendu ondoren geratzen den zerrenda itzuliko du. Zenbakia zerrendaren luzera baino handiagoa baldin bada, zerrenda hutsa itzuliko da. Zenbakia zero edo negatiboa baldin bada, zerrenda osoa itzuliko da. Zenbakiak Int motakoa izan beharko du.

```
drop 2 [4, 6, 8] → [8]
drop 5 [4, 6, 8] → []
drop (-2) [4, 6, 8] → [4, 6, 8]
drop 0 [4, 6, 8] → [4, 6, 8]
```

dropWhile :: (t -> Bool) -> [t] -> [t]

Balio Boolearra itzultzen duen funtzio bat eta zerrenda bat emanda, funtzioak False balioa ematen dion zerrendako lehenengo elementura arteko elementuak kenduz osatutako zerrenda itzuliko du. Beraz funtzioak False ematen dion elementu bat aurkitu arte edo zerrenda bukatu arte jarraituko du.

```
dropWhile (< 3) [1, 2, 3, 4, 1, 2, 3, 4] == [3, 4, 5, 1, 2, 3]
dropWhile (< 9) [1, 2, 3] == []
dropWhile (< 0) [1, 2, 3] == [1, 2, 3]
```

maximum :: Ord t => [t] -> t
Zerrendako balio handiena itzuliko du. Zerrendako elementuentzat ordenak definituta egon beharko du (txikiagoa izatea, handiagoa, eta abar).

maximum [4, 9, 9, 8] → 9 maximum [] → error

minimum :: Ord t => [t] -> t
Zerrendako balio txikiena itzuliko du. Zerrendako elementuentzat ordenak definituta egon beharko du (txikiagoa izatea, handiagoa, eta abar).

minimum [4, 9, 9, 8] → 4 minimum [] → error

concat :: [[t]] -> [t]
Zerrendez osatutako zerrenda bat emanda, zerrenda denak elkartuz zerrenda bakarra itzuliko du.

concat [[1, 2, 3], [4], [], [5, 6]] → [1, 2, 3, 4, 5, 6]

and :: [Bool] -> Bool
Balio Boolearrez osatutako zerrenda bat emanda, denak True badira, True eta bestela False itzuliko du.

```
and [4 > 1, True, 5 == 5] → True
and [4 < 1, True, 5 == 5] → False
```

or :: [Bool] -> Bool
Balio Boolearrez osatutako zerrenda bat emanda, gutxienez bat True bada, True eta bestela, False itzuliko du.

```
or [4 > 1, True, 5 == 5] → True
or [4 < 1, True, 5 == 5] → True
```

map :: (t1 -> t2) -> [t1] -> [t2]
Funtzio bat eta zerrenda bat emanda, funtzioa zerrendako elementu bakoitzari aplikatuz osatzen den zerrenda berria itzuliko du.

map (>5) [3, 8, 5, 9] → [False, True, False, True]

map (+2) [3, 8, 5, 9] → [5, 10, 7, 11]

any :: (t -> Bool) -> [t] -> Bool
Balio Boolearra itzultzen duen funtzio bat eta zerrenda bat emanda, funtzioak zerrendako elementuren batentzat True itzultzen badu, orduan any-k ere True itzuliko du eta bestela False itzuliko du.

any (>5) [3, 8, 5, 9] → True
any (>10) [3, 8, 5, 9] → False

all :: (t -> Bool) -> [t] -> Bool
Balio Boolearra itzultzen duen funtzio bat eta zerrenda bat emanda, funtzioak zerrendako elementu denentzat True itzultzen badu, orduan all-ek ere True itzuliko du eta bestela False itzuliko du.

all (>5) [3, 8, 5, 9] → False
all (>2) [3, 8, 5, 9] → True

zip :: [t1] -> [t2] -> [(t1, t2)]
Bi zerrenda emanda, posizio berean dauden elementuekin eratutako bikoteez osatutako zerrenda itzuliko du. Zerrenda bat bestea baino laburragoa bada, bikote-eraketa zerrenda laburrena bukatzean bukatuko da.

zip [1, 2, 3] [4, 5, 6] → [(1, 4), (2, 5), (3, 6)]
zip [1, 2] [4, 5, 6] → [(1, 4), (2, 5)]

Azpimarratzekoa da zip [1, 2, 3] [4, 5, 6] eta [(x, y) | x <- [1, 2, 3], y <- [4, 5, 6]] zerrenda desberdinak direla, izan ere azken hau honako zerrenda hau da:
[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]

zip3 :: [t1] -> [t2] -> [t3] -> [(t1, t2, t3)]
Hiru zerrenda emanda, posizio bereko elementuez eratutako hirukoteen zerrenda itzuliko du.

zip3 [1, 2, 3] [4, 5, 6] [7, 8, 9]→ [(1, 4, 7), (2, 5, 8), (3, 6, 9)]
zip3 [1, 2] [4, 5, 6] [7, 8, 9]→ [(1, 4, 7), (2, 5, 8)]

7.8.2. Data.List modulukoak diren funtzioak

Atal honetan aipatuko diren funtzioak Data.List modulukoak dira eta erabili ahal izateko honako agindu hau ipini beharko da:

import Data.List

Data.List modulua Haskell lengoaian aurredefinituta dagoen modulu bat da eta agindua lerroa ipintzea nahikoa da bertako funtzioak erabiltzeko.

genericLength :: Num i => [t] -> i
Zerrendaren luzera (elementu kopurua) itzuliko du zenbakizko edozein i mota erabiliz.

genericLength [4, 6, 8] → 3 genericLength [] → 0

genericTake :: Integral t1 => t1 -> [t2] -> [t2] Zenbaki bat eta zerrenda bat emanda, zerrendaren hasieratik zenbakiak adierazten duen adina elementu hartuz osatutako zerrenda itzuliko du. Zenbakia zerrendaren luzera baino handiagoa baldin bada, zerrenda osoa itzuliko da. Zenbakia zero edo negatiboa baldin bada, zerrenda hutsa itzuliko da. Zenbakiak Int edo Integer motakoa izan beharko du, hau da, Integral erakoa.

genericTake 2 [4, 6, 8] → [4, 6]
genericTake 5 [4, 6, 8] → [4, 6, 8]
genericTake (-2) [4, 6, 8] → []
genericTake 0 [4, 6, 8] → []

genericDrop :: Integral t1 => t1 -> [t2] -> [t2]
Zenbaki bat eta zerrenda bat emanda, zerrendaren hasieratik zenbakiak adierazten duen adina elementu kendu ondoren geratzen den zerrenda itzuliko du. Zenbakia zerrendaren luzera baino handiagoa bada, zerrenda hutsa itzuliko da. Zenbakia zero edo negatiboa baldin bada, zerrenda osoa itzuliko da. Zenbakiak Int edo Integer motakoa izan behar du, hau da Integral erakoa.

genericDrop 2 [4, 6, 8] → [8]
genericDrop 5 [4, 6, 8] → []
genericDrop (-2) [4, 6, 8] → [4, 6, 8]
genericDrop 0 [4, 6, 8] → [4, 6, 8]

7.8.3. Bereziki String motarentzat Prelude moduluan dauden funtzioak

Preluden, bereziki String motarentzat definituak izan diren funtzio batzuk daude. Funtzio horietako batzuk honako hauek dira:

- **lines**: Karaktere-kate bat emanda, karaktere kate hori osatzen duten lerroez eratutako zerrenda itzuliko du. Lerro-jauzia '\n' espresioaren bidez adierazten da. Funtzio honen mota honako hau da:

```
lines :: String -> [String]
```


Adibidez,

```
lines "aaa\nbbbb\nccc"
```

kasuan erantzuna honako hau da:

```
["aaa","bbbb","ccc"]
```

- **unlines:** Karaktere-katez osatutako zerrenda bat emanda, zerrenda horretako kate bakoitza lerro bat dela kontsideratuz osatzen den katea itzuliko du. Beraz, zerrenda osatzen duten kateak elkartu egingo dira '\n' lerro-jauzia tartekatuz. Funtzio honen mota honako hau da:

```
unlines :: [String] -> String
```

Adibidez,

```
unlines ["aaa","bbbb","ccc"]
```

kasuan erantzuna honako hau izango da:

```
"aaa\nbbbb\nccc\n"
```

- **words**: Karaktere-kate bat emanda, karaktere kate hori osatzen duten hitzez eratutako zerrenda itzuliko du. Zuriuneak eta lerro-jauziak ('\n') dira hitzen mugatzailetzat hartuko direnak. Funtzio honen mota honako hau da:

```
words :: String -> [String]
```

Adibidez,

```
words "aaa bbbb\nccc    dd"
```

kasuan erantzuna honako hau izango da:

```
["aaa","bbbb","ccc","dd"]
```

Adibidean ikus daiteke hitzak zuriune batez baino gehiagoz bereizita egon daitezkeela. Zuriunez osatutako bloke osoak bi hitz bereizteko balio du.

- **unwords**: Karaktere-katez osatutako zerrenda bat emanda, zerrenda horretako kate bakoitza hitz bat dela kontsideratuz osatzen den katea itzuliko du. Beraz, zerrenda osatzen duten kateak elkartu egingo dira zuriunea tartekatuz. Funtzio honen mota honako hau da:

```
unwords :: [String] -> String
```

Adibidez,

```
unwords ["aaa","bbbb","ccc   ","d   d"]
```

kasuan erantzuna honako hau izango da:

```
"aaa bbbb ccc    d   d"
```


Adibidean ikusten da sarrerako zerrendako hitzek zuriuneak baldin badituzte, zuriune horiek mantendu egingo direla.

"aaa bbbb ccc d d" kateari words aplikatzen badiogu, ez dugu lortuko abiapuntuko zerrenda, beste zerrenda hau lortuko dugu:

```
["aaa","bbbb","ccc","d","d"]
```

Eta hor hitz bat gehiago dago.