## მუშაობის საფუძვლები

### Haskell გარემო

Haskell-ის შე­სას­წავ­ლად, უპირ­ვე­ლეს ყოვ­ლი­სა, სა­ჭი­როა კომ­პი­უ­ტერ­ზე დად­გეს ინ­ტერ­პრე­ტა­ტო­რი, ანუ პროგ­რა­მა – ტრან­სლა­ტო­რი, რო­მე­ლიც Haskell-ის კოდს გა­და­იყ­ვანს ორო­ბით კოდ­ში და ამ კოდს შე­ას­რუ­ლებს. Haskell მრა­ვალ­მხრივ რე­ა­ლი­ზე­ბუ­ლი­ა. მათ­გან ორი – ყვე­ლა­ზე გავ­რცე­ლე­ბუ­ლი­ა. Hugs – ინ­ტერ­პრე­ტა­ტო­რი ძი­რი­თა­დად სას­წავ­ლო მიზ­ნით გა­მო­ი­ყე­ნე­ბა. რე­ა­ლუ­რი პროგ­რა­მე­ბის­თვის Glasgow Haskell Compiler (GHCi) კომ­პი­ლე­რი უფ­რო პო­პუ­ლა­რუ­ლი­ა. ის კომ­პი­ლი­რებს მან­ქა­ნურ კოდ­ში, მხარს უჭერს პა­რა­ლე­ლურ დაპ­როგ­რა­მე­ბას და აქვს პროგ­რა­მის ანა­ლი­ზი­სა და გა­მარ­თვის სა­შუ­ა­ლე­ბე­ბი. აქე­დან გა­მომ­დი­ნა­რე, ჩვენ მი­ერ შემ­დგომ GHCi იქ­ნე­ბა გა­მო­ყე­ნე­ბუ­ლი.

შემ­დეგ მი­სა­მარ­თებ­ზე მოყ­ვა­ნი­ლია Haskell-ის ინ­ტერ­პრე­ტა­ტო­რის, სა­ხე­ლად GHCi-ს დო­კუ­მენ­ტა­ცია და და­ინ­სტა­ლი­რე­ბის ინ­სტრუქ­ცი­ა:

<http://docs.haskellstack.org/en/stable/README/>

<https://github.com/bitemyapp/learnhaskell>

### Haskell-ის ინტერაქტიული კოდი. REPL-ის გამოყენება

Haskell-ი გვაძ­ლევს კოდ­თან მუ­შა­ო­ბის ორ გზას: პირ­ვე­ლია კო­დის შე­ტა­ნა უშუ­ა­ლოდ GHCi-ის ინ­ტე­რაქ­ტი­ულ გა­რე­მო­ში ანუ REPL; მე­ო­რეა კო­დის შე­ტა­ნა საწყ­ის ფა­ილ­ში, შე­ნახ­ვა და შემ­დეგ საწყ­ი­სი ფა­ი­ლის ჩატ­ვირ­თვა GHCi-ში. ჩვენ შე­ვე­ხე­ბით ორი­ვე მე­თოდს.

REPL არის შე­მოკ­ლე­ბა შემ­დე­გი ფრა­ზი­სა: *read-eval-print loop* (ციკ­ლი: წა­კითხ­ვა-­მნიშ­ვნე­ლო­ბის გა­მოთ­ვლა-­და­ბეჭ­დვა). REPL არის ინ­ტე­რაქ­ტი­უ­ლი დაპ­როგ­რა­მე­ბის გა­რე­მო, სა­დაც შეგ­ვიძ­ლია შე­ვი­ტა­ნოთ კო­დი, რომ­ლის მნიშ­ვნე­ლო­ბაც გა­მო­ით­ვლე­ბა და და­ვი­ნა­ხავთ შე­დეგს. ეს პრო­ცე­სი პირ­ვე­ლად გაჩ­ნდა ენა ლის­პში, თუმ­ცა ამ­ჟა­მად სა­ერ­თოა დაპ­როგ­რა­მე­ბის თა­ნა­მედ­რო­ვე ენე­ბის­თვის, მათ შო­რის Haskell-ის­თვის.

ინ­ტერ­პრე­ტა­ტო­რის და­ყე­ნე­ბი­სა და გაშ­ვე­ბის შემ­დეგ ჩვენ და­ვი­ნა­ხავთ შეტყ­ო­ბი­ნე­ბას ვერ­სი­ის შე­სა­ხებ, გან­ცხა­დე­ბას (თუ გსურთ დახ­მა­რე­ბა, აკ­რი­ფეთ სიმ­ბო­ლო ?) და „მოწვევას სამუშაოდ“ (სტრი­ქო­ნი Prelude>):

|  |
| --- |
| GHCi, version 7.10.3 : http://www.haskell.org/ghc/ :? for help  Prelude> |

რა თქმა უნ­და, ვერ­სია შე­იძ­ლე­ბა იყოს სხვა.

ბრძა­ნე­ბა­თა სტრი­ქონ­ში სიტყ­ვა Prelude მი­უ­თი­თებს, რომ ჩატ­ვირ­თუ­ლია და მზად არის გა­მო­სა­ყე­ნებ­ლად სტან­დარ­ტუ­ლი ბიბ­ლი­ო­თე­კა Prelude. რო­ცა ჩავ­ტვირ­თავთ სხვა მო­დუ­ლებს ან საწყ­ის ფა­ი­ლებს, მა­შინ ისი­ნი აი­სა­ხე­ბა დი­ა­ლო­გურ ფან­ჯა­რა­ში. შე­საძ­ლე­ბე­ლია Prelude-ის გა­მორ­თვაც, რა­საც შემ­დეგ შე­ვე­ხე­ბით.

მო­დულს Prelude ზოგ­ჯერ უწო­დე­ბენ „სტანდარტულ პრელუდიას“, ვი­ნა­ი­დან მი­სი ში­ნა­არ­სი გა­ნი­საზღ­ვრე­ბა Haskell 98-ის სტან­დარ­ტით.

მო­საწ­ვე­ვის პა­სუ­ხად შე­ვი­ტა­ნოთ რამ­დე­ნი­მე მარ­ტი­ვი არით­მე­ტი­კუ­ლი გა­მო­სა­ხუ­ლე­ბა:

|  |
| --- |
| Prelude> 2 + 2  4  Prelude> 7 < 9  True  Prelude> 10 ^ 2  100 |

დავ­წე­როთ ჩვე­ნი პირ­ვე­ლი პროგ­რა­მა Haskell-ზე, რო­მელ­საც ეკ­რან­ზე გა­მო­აქვს მი­სა­სალ­მე­ბე­ლი სტრი­ქო­ნი. ფუნ­ქცი­ას და­ვარ­ქვათ sayHello და გა­დავ­ცეთ ერ­თი სტრი­ქო­ნუ­ლი არ­გუ­მენ­ტი. ფუნ­ქცი­ის სა­ხელ­თან მი­სი გან­მარ­ტე­ბის და­სა­კავ­ში­რებ­ლად გა­მო­ვი­ყე­ნოთ ოპე­რა­ტო­რი let (სა­ხელ­თან და­კავ­ში­რე­ბის ოპე­რა­ტო­რი). REPL გა­რე­მო­ში ავ­კრი­ფოთ ფუნ­ქცი­ის გან­მარ­ტე­ბა:

|  |
| --- |
| Prelude> let sayHello x = putStrLn ("Hello, " ++ x ++ "!") |

მა­გა­ლით­ში გა­მო­ყე­ნე­ბუ­ლი გვაქვს putStrLn, რო­მე­ლიც არის სტრი­ქო­ნის ეკ­რან­ზე გა­მო­ტა­ნის ფორ­მა­ტი­რე­ბუ­ლი ოპე­რა­ტო­რი. მა­გა­ლი­თად:

|  |
| --- |
| Prelude> putStrLn "Here's a newline -->\n <-- See?"  Here's a newline -->  <-- See? |

ახ­ლა მო­ვახ­დი­ნოთ ფუნ­ქცი­ის გა­მო­ძა­ხე­ბა არ­გუ­მენ­ტით "Haskell" და ვნა­ხოთ შე­დე­გი:

|  |
| --- |
| Prelude> sayHello "Haskell"  Hello, Haskell! |

აქ გა­მო­ყე­ნე­ბუ­ლია ოპე­რა­ტო­რი ++ კონ­კა­ტე­ნა­ცი­ა, რო­მე­ლიც ახ­დენს სტრი­ქო­ნე­ბის შერ­წყმას.

GHCi-დან გამოსასვლელად გამოიყენეთ ბრძანება :quit ან :q.

### მუშაობა საწყის ფაილებთან

მუ­შა­ო­ბა საწყ­ის ფა­ი­ლებ­თან ისე­ვე მო­სა­ხერ­ხე­ბე­ლი­ა, რო­გორც მუ­შა­ო­ბა REPL-ის გა­რე­მო­ში. ფა­ი­ლის გა­ფარ­თო­ე­ბა სა­ვალ­დე­ბუ­ლოა იყოს .hs, რო­მე­ლიც მი­უ­თი­თებს, რომ ეს არის საწყ­ი­სი კო­დი Haskell-ზე. ამ რე­ჟიმს მა­შინ ვი­ყე­ნებთ, რო­ცა სა­ჭი­როა კო­დის თან­და­თა­ნო­ბით აგე­ბა. კო­დის შექ­მნის შემ­დეგ ამ ფა­ილს ჩავ­ტვირ­თავთ REPL-ში და შემ­დეგ ძი­რი­თა­დი პრო­ცე­სი და­დის ინ­ტე­რაქ­ტი­ულ პრო­ცეს­ზე: კო­დის გაშ­ვე­ბა, მო­დი­ფი­კა­ცია და ტეს­ტი­რე­ბა.

საწყ­ი­სი ფა­ი­ლის შეს­რუ­ლე­ბაც შე­საძ­ლე­ბე­ლია ორი გზით: ან გა­ვუშ­ვათ საწყ­ი­სი კო­დის ფა­ი­ლი ან REPL-ში გავ­ხსნათ იგი. პირ­ველ შემ­თხვე­ვა­ში გა­იხ­სნე­ბა gnci.exe ინ­ტერ­პრე­ტა­ტო­რის ფან­ჯა­რა, სა­დაც უკ­ვე ჩატ­ვირ­თუ­ლი იქ­ნე­ბა ფა­ი­ლის ში­ნა­არ­სი. მე­ო­რე შემ­თხვე­ვა­ში კი გავ­ხსნით REPL-ს და სპე­ცი­ა­ლუ­რი დი­რექ­ტი­ვის (:load) გა­მო­ყე­ნე­ბით ჩავ­ტვირ­თავთ ფა­ილს.

გავ­ხსნათ ახა­ლი Text Documen. სა­ჭი­როა ფა­ილს და­ერ­ქვას სა­ხე­ლი, მა­გა­ლი­თად, test.hs. შე­ვი­ტა­ნოთ ამ ფა­ილ­ში შემ­დე­გი კო­დი და შე­ვი­ნა­ხოთ იგი იმა­ვე სა­ხე­ლით (test.hs):

|  |
| --- |
| sayHello :: String -> IO ()  sayHello x = putStrLn ("Hello, " ++ x ++ "!") |

აქ ნიშანი „::“ აღნიშნავს ტიპს, ანუ ჩვენ ვამბობთ, რომ „აქვს ტიპი“. ასე რომ, sayHello-ს აქვს ტი­პი String -> IO (). ტი­პებ­ზე ჩვენ შემ­დგომ ვი­სა­უბ­რებთ. შევ­ნიშ­ნოთ, რომ ეს სტრი­ქო­ნი არ არის აუ­ცი­ლე­ბე­ლი, ვი­ნა­ი­დან Haskell თვი­თონ შეძ­ლებს ამო­იც­ნოს ჩვენ მი­ერ გან­საზღ­ვრუ­ლი ფუნ­ქცი­ის ტი­პი.

გავხსნათ WinGHCi და ჩავწეროთ შემდეგი ბრძანებები:

|  |
| --- |
| Prelude> :load test.hs  \*Main> sayHello "Haskell"  Hello, Haskell!  \*Main> |

პირ­ვე­ლი ბრძა­ნე­ბის :load ჩატ­ვირ­თავს ფა­ილს test.hs, რის შემ­დე­გად ფუნ­ქცია sayHello უკ­ვე ჩანს REPL-ში, ამი­ტო­მაც შე­გიძ­ლი­ათ გა­დას­ცეთ ნე­ბის­მი­ე­რი სტრი­ქო­ნი­-არ­გუ­მენ­ტი (მა­გა­ლი­თად,"Haskell") და ნა­ხოთ შე­დე­გი. და­აკ­ვირ­დით, რომ ფა­ი­ლის ჩატ­ვირ­თვის შემ­დეგ მო­საწ­ვე­ვი სტრი­ქო­ნი შე­იც­ვა­ლა. თუ გა­მო­ი­ყე­ნებთ GHCi-ის ბრძა­ნე­ბა :m-ს, რო­მე­ლიც არის ბრძა­ნე­ბა :module-ის შე­მოკ­ლე­ბა, მა­შინ დაბ­რუნ­დე­ბა Prelude> და ჩატ­ვირ­თუ­ლი ფა­ი­ლი უკ­ვე აღარ იქ­ნე­ბა REPL-ის ხედ­ვის წვდო­მის არე­ში:

|  |
| --- |
| \*Main> :m  Prelude> sayHello "Haskell"  <interactive>:12:1: Not in scope: ‘sayHello’  (0.00 secs, 0 bytes)  Prelude> |