**Haskell გარემო**

Haskell — ეს არის ენა, რომელსაც მრავალი რეალიზაცია აქვს. მათგან ორი – ყველაზე გავრცელებულია. Hugs — ინტერპრეტატორი, ძირითადად სასწავლო მიზნით გამოიყენება. რეალური პროგრამებისთვის Glasgow Haskell Compiler (GHC) კომპილერი უფრო პოპულარულია. ის კომპილირებს მანქანურ კოდში, მხარს უჭერს პარალელურ პროგრამირებას და აქვს პროგრამის ანალიზისა და გამართვის საშუალებები. აქედან გამომდინარე, ჩვენს მიერ შემდგომში GHC იქნება გამოყენებული.

GHC პლათფორმის ჩასატვირთად, ეწვიეთ გვერდს [the GHC download page](http://www.haskell.org/ghc/download.html).

დავიწყოთ მუშაობა ghci ინტერპრეტატორში.

ინტერაქტიული ინტერპრეტატორი არის პროგრამა სახელით ghci. ის იძლევა საშუალებას შევიტანოთ და გამოვთვალოთ ჰასკელის გამოსახულებები, გამოვიკვლიოთ მოდულები და გავმართოთ ჩვენი კოდი.

ghciგაშვების შემდეგ გამოდის შეტყობინება ვერსიის შესახებ, განცხადებები და მოწვევა სამუშაოდ – სტრიქონი Prelude>

$ ghci

GHCi, version 7.0.3: http://www.haskell.org/ghc/ :? for help

Loading package base ... linking ... done.

Prelude>

ბრძანებათა სტრიქონში სიტყვა Prelude მიუთითებს, რომ ჩატვირთულია სტანდარტული ბიბლიოთეკა Prelude და მზად არის გამოსაყენებლად. როცა ჩავტვირთავთ სხვა მოდულებს ან საწყის ფაილებს, მაშინ ისინი აისახება დიალოგურ ფანჯარაში.

|  |  |
| --- | --- |
| [Tip] | ცნობების მიღება |
| თუ ghci–ის ბრძანებათა სტრიქონში შეიტანთ :?  დაიბეჭდება საცნობარო ინფორმაცია დირექტივების შესახებ. |

მოდულს Prelude ზოგჯერ უწოდებენ „სტანდარტულ პრელუდიას“, ვინაიდან მისი შინაარსი განისაზღვრება Haskell 98–ის სტანდარტით. სიმოკლის მიზნით მოცემულ სახელმძღვანელოში გამოყენებულია მოწვევა ghci>. ამის განხორციელება ხდება დირექტივით :set promp, როგორც მოცემულია ქვემოთ:

Prelude> :set prompt "ghci> "

ghci> :module + Data.Ratio

Prelude ყოველთვის არის მიღწევადი. სხვა მოდულების გამოსაყენებლად საჭიროა ჩატვირთოთ ბრძანებით :module

ghci> :module + Data.Ratio

მოდული Data.Ratio გვაძლევს რაციონალურ რიცხვებთან (წილადებთან) მუშაობის საშუალებას.

ghci **–ის გამოყენება გამოთვლებში**

მარტივი არითმეტიკა

შევიტანოთ გამოსახულებები, რათა ვნახოთ, როგორ გამოთვლის მათ ინტერპრეტატორი. გამოსახულებებს ჩავწერთ *ინფიქსური ფორმით* (in *infix* form)**,** სადაც ოპერატორი მოთავსებულია ოპერანდებს შორის.

ghci> 2 + 2

4

ghci> 31337 \* 101

3165037

ghci> 7.0 / 2.0

3.5

ჩანაწერის ინფიქსური ფორმის გარდა შესაძლებელია *პრეფიქსული ფორმის* (in *prefix* form) გამოყენება. ამისთვის საჭიროა ოპერატორი ჩაისვას მრგვალ ფჩხილებში.

ghci> 2 + 2

4

ghci> (+) 2 2

4

Haskell–ში არის მთელი და მცოცავმძიმიანი რიცხვების ცნება. ნიშანი (^) –არის მთელი რიცხვის ხარისხში აყვანის ოპერაცია.

ghci> 313 ^ 15

27112218957718876716220410905036741257

**უარყოფითი რიცხვების ჩაწერა**

Haskell–ში ზოგიერთ შემთხვევაში საჭიროა უარყოფითი რიცხვების ფრჩხილებში მოთავსება. ეს ეხება რთულ გამოსახულებებს.

ჩავწეროთ უარყოფითი რიცხვი:

ghci> -3

-3

ოპერაცია „-“ არის უნარული. არ შეიძლება მისი არევა ინფიქსურ ოპერატორთან. არ შეიძლება:

ghci> 2 + -3

<interactive>:1:0:

precedence parsing error

cannot mix `(+)' [infixl 6] and prefix `-' [infixl 6] in the same infix expression

საჭიროა „უნარული მინუსი“ ჩაისვას მრგვალ ფრჩხილებში:

ghci> 2 + (-3)

-1

ghci> 3 + (-(13 \* 37))

-478

ეს საშუალებას გვაძლევს ავიცილოთ სინტაქსური განუსაზღვრელობა. როცა Haskell–ში ვიყენებთ ფუნქციას, ვწერთ ფუნქციის სახელს, შემდეგ არგუმენტს, მაგალითად, f 3. თუ უარყოფით რიცხვებს არ ჩავსვავდით ფრჩხილებში, მაშინ გამოსახულება f-3 შეიძლება გაგვეგო როგორც „f–ის გამოყენება მთელ -3 რიცხვთან“ ან „გამოვაკლოთ რიცხვი 3 f ცვლადს“.

დადებითი რიცხვების გამრავლება:

ghci> 2\*3

6

უარყოფით რიცხვზე გამრავლება (თუ არ ჩავსვავთ უარყოფით რიცხვს ფრჩხილებში) იწვევს შეცდომას:

ghci> 2\*-3

<interactive>:1:1: Not in scope: `\*-'

ინტერპრეტატორი მიმდევრობით ნიშნებს აღიქვავს როგორც ერთ ოპერატორს და უთითებს შეცდომაზე: „არ არსებობს ოპერატორი \*-“. საჭიროა:

ghci> 2\*(-3)

-6

**ბულის ლოგიკა, ოპერატორები და გამოსახულებების შედარება**

Haskell–ში ლოგიკური მნიშვნელობებია True და False. დიდი ასოთი დაწყება აუცილებელია. ლოგიკური ოპერაციებია: (&&) — ლოგიკური «და», — (||) ლოგიკური «ან».

ghci> True && False

False

ghci> False || True

True

ზოგიერთ ენაში რიცხვი 0 ასოცირდება როგორც ლოგიკური False, ხოლო არანულოვანი მნიშვნელობა – ლოგიკური True. Haskell–ში ასე არ არის:

ghci> True && 1

<interactive>:1:8:

No instance for (Num Bool)

arising from the literal `1' at <interactive>:1:8

Possible fix: add an instance declaration for (Num Bool)

In the second argument of `(&&)', namely `1'

In the expression: True && 1

In the definition of `it': it = True && 1

ამ შეცდომას იწვევს ის, რომ ლოგიკური ტიპი არ არის რიცხვითი Num ტიპი.

აი ზოგიერთი ტიპიური შეტყობინება შეცდომის შესახებ:

* "No instance for (Num Bool)" გვეუბნება, რომ ინტერპრეტატორი ghci ცდილობს გარდაქმნას მნისვნელობა 1 როგორც Bool ტიპი, მაგრამ ეს არ შეიძლება.
* "arising from the literal `1'" მიუთითებს, რომ მთელი რიცხვის 1–ის გამოყენება იწვევს პრობლემას.
* "In the definition of `it'" – ghci მიუთითებს, რომ it გადასაზღვრულია (შემდეგში განვმართავთ).

შედარების ოპერატორები Haskell–ში მსგავსია c ენის ოპერატორების:

ghci> 1 == 1

True

ghci> 2 < 3

True

ghci> 4 >= 3.99

True

მხოლოდ ოპერატორი „არ უდრის“ განსხვავდება c ენის ოპერატორისგან (!=). იგი Haskell–ში აღინიშნება /=:

ghci> 2 /= 3

True

გარდა ამისა, C-ის ტიპის ენებში გამოიყენება ლოგიკური უარყოფისთვის ნიშანი !, ხოლო Haskell–ში ფუნქცია not .

ghci> not True

False

**ასოციურობა**

როგორც სხვა ენებში, Haskell–ის ინფიქსური ოპერატორებისთვისაც განისაზღვრება პრიორიტეტები. გამოსახულების ცხადი დაჯგუფებისთვის გამოიყენება მრგვალი ფრჩხილები. ვინაიდან გამრავლებას უფრო მაღალი პრიორიტეტი აქვს, შემდეგი ორი ჩანაწერი ერთიდაიგივეა:

ghci> 1 + (4 \* 4)

17

ghci> 1 + 4 \* 4

17

Haskell–ში ოპერატორებს ენიჭება რიცხვითი მნიშვნელობები 1–დან (ყველაზე პატარა პრიორიტეტი) 9–მდე. იმის გასაგებად, თუ რა პრიორიტეტი აქვს ოპერატორს, შეიძლება გამოვიყენოთ :info –ინფორმაცია ბრძანების შესახებ:

ghci> :info (+)

class (Eq a, Show a) => Num a where

(+) :: a -> a -> a

...

-- Defined in GHC.Num

infixl 6 +

ghci> :info (\*)

class (Eq a, Show a) => Num a where

...

(\*) :: a -> a -> a

...

-- Defined in GHC.Num

infixl 7 \*

ინფორმაცია "infixl 6 +" მიუთითებს, რომ ოპერატორს (+) აქვს პრიორიტეტი 6. (სხვა მონაცემებს შემდეგ განვიხილავთ). "infixl 7 \*" განსაზღვრავს, რომ ოპერატორს (\*) აქვს პრიორიტეტი 7. ამის გამო ვხედავთ, რომ 1 + 4 \* 4 გამოითვლება როგორც 1 + (4 \* 4) და არა, როგორც (1 + 4) \* 4.

Haskell–ში განსაზღვრულია ასევე ოპერატორების ასოციურობა–ოპერატორის მრავალჯერადი გამოყენების შესაძლებლობა მარცხნიდან მარჯვნივ, ან პირიქით. (+) და (\*) მარცხნიდან ასოციურია და აღინიშნება როგორც infixl. მარჯვნიდან მარცხნივ ასოციურობა აღინიშნება infixr. ასეთია, მთელი რიცხვის ახარისხების ოპერაცია.

ghci> :info (^)

(^) :: (Num a, Integral b) => a -> b -> a -- Defined in GHC.Real

infixr 8 ^

განუსაზღვრელი მნიშვნელობები

Haskell–ის სტანდარტულ ბიბლიოთეკაში prelude განსაზღვრულია მათემატიკური კონსტანტები:

ghci> pi

3.141592653589793

მაგრამ ყველა კონსტანტა, რა თქმა უნდა, prelude–ში განმარტებული არაა. მაგალითად, ეილერის რიცხვი

ghci> e

<interactive>:1:0: Not in scope: `e'

საჭიროა ჩვენვე განვსაზღვროთ იგი.ghci–ში არის let კონსტუქცია, რომლის საშუალებითაც შეიძლება e–ს დროებითი განსაზღვრა:

ghci> let e = exp 1

exp 1 არის ჩვენთვის Haskell ფუნქციის გამოყენების პირველი მაგალითი. მრგვალი ფრჩხილების გამოყენება არგუმენტისთვის საჭირო არ არის. ასე განსაზღვრული e შეიძლება გამოყენებული იყოს არითმეტიკულ გამოსახულებებში. ოპერატორი ახარისხება (^) გამოიყენება მხოლოდ მთელი რიცხვებისთვის, ნამდვილი რიცხვებისთვის გამოიყენება (\*\*). მაგალითი:

ghci> (e \*\* pi) - pi

19.99909997918947

ეს სინტაქსი დამოკიდებულია ghci–ის კონკრეტულ რეალიზაციაზე!

|  |  |
| --- | --- |
| [Tip] | სად შეიძლება მოვიძიოთ დამატებითი ინფორმაცია: |
| Unix სისტემებისთვის [ბიბლიოთეკები GNU readline](http://www.microsofttranslator.com/bv.aspx?from=&to=ru&a=http%3A%2F%2Ftiswww.case.edu%2Fphp%2Fchet%2Freadline%2Frltop.html%23Documentation). Windows–ისთვის, [ბრძანება **doskey**](http://www.microsofttranslator.com/bv.aspx?from=&to=ru&a=http%3A%2F%2Fwww.microsoft.com%2Fresources%2Fdocumentation%2Fwindows%2Fxp%2Fall%2Fproddocs%2Fen-us%2Fdoskey.mspx) |

**შესავალი ტიპებში**

Haskell-ში მოითხოვება, რომ ტიპის სახელები დაიწყოს დიდი ასოთი.

ინტერპრეტატორ ghci-ში არსებობს მრძანება :set, რომელიც ნებას გვაძლევს შევცვალოთ რამდენიმე წინასწარ დაყენებული პარამეტრი. :set +t სტრიქონით მოვითხოვთ დაიბეჭდოს ინფორმაცია ტიპების შესახებ.

ghci> :set +t

ghci> 'c'

'c'

it :: Char

ghci> "foo"

"foo"

it :: [Char]

გამოსახულების მნიშვნელობის შემდეგ ტიპის დაბეჭდვა ძალზე სასარგებლოა.

ყურადღება გავამახვილოთ:

* სახელი it არის სპეციალური ცვლადი, რომელშიც ghci ინახავს ბოლო გამოსახულების გამოთვლილ მნიშვნელობას.
* x :: ფორმის ტექსტს აქვს ასეთი შინაარსი: „გამოსახულება x-ს აქვს ტიპი y “.
* სტრიქონს "foo" აქვს ტიპი[Char]. სიტყვა string ხშირად გამოიყენება ნაცვლად [Char]-ისა. ისინი სინონიმებია.

|  |  |
| --- | --- |
| [Tip] | it -ის შესახებ |
| ცვლადი it ghci-ის მოსახერხებელი სტრუქტურაა, რომელიც გვაძლევს საშუალებას გამოყენებული იყოს ბოლო გამოთვლილი გამოსახულების მნიშვნელობა.  ghci> "foo"  "foo"  it :: [Char]  ghci> it ++ "bar"  "foobar"  it :: [Char]  გამოსახულების გამოთვლისას, თუ მოხდა შეცდომა, მაშინ ghci არ ცვლის it-ის მნიშვნელობას.  ghci> it  "foobar"  it :: [Char]  ghci> it ++ 3  <interactive>:1:6:  No instance for (Num [Char])  arising from the literal `3' at <interactive>:1:6  Possible fix: add an instance declaration for (Num [Char])  In the second argument of `(++)', namely `3'  In the expression: it ++ 3  In the definition of `it': it = it ++ 3  ghci> it  "foobar"  it :: [Char]  ghci> it ++ "baz"  "foobarbaz"  it :: [Char]  კიდევ რამდენიმე სიტყვა ტიპებზე. ჩვენ უკვე ვიცნობთ ახარისხების ოპერაციას. |

ghci> 7 ^ 80

40536215597144386832065866109016673800875222251012083746192454448001

it :: Integer

Haskell მთელი რიცხვების ტიპია Integer. Integer ტიპის მნიშვნელობია შემოსაზღვრულია მხოლოდ თქვენი სისტემის მეხსიერებით.

რაციონალური რიცხვების გამოსაყენებლად საჭიროა ავკრიფოთ სტრიქონი :m +Data.Ratio. რაციონალური რიცხვების მისაღებად უნდა გამოვიყენოთ ოპერატორი(%), რომლის მარცხნივ უნდა იყოს მრიცხველი, მარჯვნივ - მნიშვნელი.

ghci> :m +Data.Ratio

ghci> 11 % 29

11%29

it :: Ratio Integer

მოხერხებულობისთვის ghci იძლევა საშუალებას შემოკლდეს ბრძანებები. ასე, რომ მოდულის ჩასატვირთად ჩვენ ვწერთ : m ნაცვლად :module.

ვხვდებით, რომ წილადებში მრიცხველს და მნიშვნელს აქვს მთელი ტიპი. ჩვენ თუ შევეცდებით მთელის ნაცვლად სხვა ტიპის მნიშვნელობის აღებას, დაფიქსირდება შეცდომა:

ghci> 3.14 % 8

<interactive>:1:0:

Ambiguous type variable `t' in the constraints:

`Integral t' arising from a use of `%' at <interactive>:1:0-7

`Fractional t'

arising from the literal `3.14' at <interactive>:1:0-3

Probable fix: add a type signature that fixes these type variable(s)

ghci> 1.2 % 3.4

<interactive>:1:0:

Ambiguous type variable `t' in the constraints:

`Integral t' arising from a use of `%' at <interactive>:1:0-8

`Fractional t'

arising from the literal `3.4' at <interactive>:1:6-8

Probable fix: add a type signature that fixes these type variable(s)

თუმცა თავდაპირველად სასარგებლოა იყოს ნაჩვენები ყველა გამოსახულების ტიპი, მაგრამ ტიპებში გარკვევის შემდეგ შესაძლოა საჭირო გახდეს ამ ობციის გამორთვა. ამისთვის საჭიროა გამოვიყენოთ ბრძანება : unset.

ghci> :unset +t

ghci> 2

2

ამ შემთხვევაშიც შეგვიძლია მივიღოთ ინფორმაცია ტიპების შესახებ ghci-ის შემდეგი ბრძანებით :type.

ghci> :type 'a'

'a' :: Char

ghci> "foo"

"foo"

ghci> :type it

it :: [Char]

ბრძანება : type ბეჭდავს ნებისმიერი გამოსახულების ტიპს, it-ისაც კი.

გაარკვიეთ, რატომ აქვს შემდეგ გამოსახულებებს სხვადასხვა ტიპები:

ghci> 3 + 2

5

ghci> :type it

it :: Integer

ghci> :type 3 + 2

3 + 2 :: (Num t) => t

Haskell-ში არის რამდენიმე რიცხვითი ტიპი. ghci გამოსახულებისთვის 3 + 2, გაჩუმების პრინციპით იღებს ტიპს Integer. მაგრამ მეორე მაგალითისთვიოს, ასახელებს, რომ ტიპი არის რიცხვითი.

**სავარჯიშოები**

|  |  |
| --- | --- |
| 1**.** | დაწერეთ შემდეგი გამოსახულებები ghci–ში. შეამოწმეთ, რა ტიპი აქვთ მათ. რას აკეთებს მოცემული ფუნქციები?   * 5 + 8 * 3 \* 5 + 8 * 2 + 4 * (+) 2 4 * sqrt 16 * succ 6 * succ 7 * pred 9 * pred 8 * sin (pi / 2) * truncate pi * round 3.5 * round 3.4 * floor 3.7 * ceiling 3.3 |
|  |  |
|  |  |
|  |  |

**სიები**

სიები თავსდება ოთხკუთხა ფრჩხილებში. ელემენტები გამოიყოფა მძიმით.

ghci> [1, 2, 3]

[1,2,3]

სიის სიგრძე ნებისმიერია. ცარიელი სია აღინიშნება როგორც [].

ghci> []

[]

ghci> ["foo", "bar", "baz", "quux", "fnord", "xyzzy"]

["foo","bar","baz","quux","fnord","xyzzy"]

აუცილებელია სიის ყველა ელემენტი იყოს ერთიდაიგივე ტიპის. თუ ამ წესს დავარღვევთ, გვექნება შეტყობინება შეცდომის შესახებ:

ghci> [True, False, "testing"]

<interactive>:1:14:

Couldn't match expected type `Bool' against inferred type `[Char]'

Expected type: Bool

Inferred type: [Char]

In the expression: "testing"

In the expression: [True, False, "testing"]

სიის ელემენტებში შეიძლება გამოვიყებოთ *ჩამოთვლის ნოტაცია* (..), Haskell თვითონ შეავსებს სიას:

ghci> [1..10]

[1,2,3,4,5,6,7,8,9,10]

აქ სიმბოლო .. აღნიშნავს ჩამოთვლას. მაგალითში განხილულია ჩაკეტილი ინტერვალი, მაგრამ შესაძლებელია ღია ინტერვალების გამოყენებაც.

ჩამონათვალში შესაძლოა მითითებული იყოს ბიჯის ზომაც. პირველი ორი ელემენტის შემდეგ მოცემულია მნიშვნელობა, რომელიც ამთავრებს სიას:

ghci> [1.0,1.25..2.0]

[1.0,1.25,1.5,1.75,2.0]

ghci> [1,4..15]

[1,4,7,10,13]

ghci> [10,9..1]

[10,9,8,7,6,5,4,3,2,1]

ჩვენ შეიძლება გამოვტოვოთ ჩამონათვალის ბოლო ელემენტი. თუ ტიპს არ აქვს ბუნებრივი ზედა ზღვარი, მაშიმ მან შეიძლება მოვცეს უსასრული სია. მაგალითად, თუ ჩავწერთ [1..] ghci–ის დიალოგურ ფანჯარაში გვექნება უსასრულო ჩამონათვალი, რომელიც ხელით უნდა გავაჩეროთ. სხვა ენებში, მაგალითად, **c–**ში ასე არ არის–მოხდება გადავსების შეცდომა. უსასრულო სიები Haskell–ში ხშირად არის სასარგებლო.

|  |  |
| --- | --- |
| [Warning] | გაფრთხილდით, ნამდვილი რიცხვებისთვის .. გამოყენების დროს: |
| ghci> [1.0..1.8]  [1.0,2.0] |

**ოპერატორები სიებთან**

არსებობს სიებთან სამუშაოდ ორი ფუნქცია. პირველი, სიების გაერთიანება ოპერატორი (++) –ის გამოყენებით:

ghci> [3,1,3] ++ [3,7]

[3,1,3,3,7]

ghci> [] ++ [False,True] ++ [True]

[False,True,True]

მეორე ოპერატორია ჩასმის ოპერატორი(:), რომელიც ამატებს ელემენტს სიის თავში. მას ხშირად უწოდებენ cons (შემოკლება construct სიტყვიდან).

ghci> 1 : [2,3]

[1,2,3]

ghci> 1 : []

[1]

სიაში ბოლო ელემენტს ასე: [1,2]:3 ვერ დავამატებთ. გვექნება შეცდომა, რომ მეორე არგუმენტი უნდა იყოს სია.

**სტრიქონები და სიმბოლოები**

ტექსტური სტრიქონი მოთავსდება ორმაგ ბრჭყალებში:

ghci> "This is a string."

"This is a string."

როგორც სხვა ენებში (მაგალითად, С–ში), არსებობს მმართველი სიმბოლოების (მაგალითად, '\n','\t') გამოყენების შესაძლებლობა

დეტალური ინფორმაცია იხილეთ მისამართზე: [დანართი B,ნიშნები, სტრიქონები და გასწორების წესები](http://www.microsofttranslator.com/bv.aspx?from=&to=ru&a=http%3A%2F%2Fbook.realworldhaskell.org%2Fread%2Fcharacters-strings-and-escaping-rules.html)

ghci> **putStrLn** "Here's a newline -->\n<-- See?"

Here's a newline -->

<-- See?

ფუნქცია putStrLn გამოჰყავს სტრიქონი ეკრანზე. მასთან ტაბულაციის (\t), ახალი სტრიქონისა (\n) და ხმოვანი სიმბოლოს (\a) გამოყენება დაშვებულია.

Haskell–ში სიმბოლო ერთმაგ ბრჭყალებში ჩაისმის:

ghci> 'a'

'a'

ტექსტური სტრიქონი წარმოადგენს სიმბოლოების თანმიმდევრობას:

ghci> let a = ['l', 'o', 't', 's', ' ', 'o', 'f', ' ', 'w', 'o', 'r', 'k']

ghci> a

"lots of work"

ghci> a == "lots of work"

True

ცარიელი სტრიქონი ჩაიწერება "" და წარმოადგენენ []-ის სინონიმს.

ghci> "" == []

True

რადგან სტრიქონი არის სიმბოლოების სია, ამიტომ ახალი სტრიქონის ასაგებად შეიძლება გამოვიყენოთ ოპერატორები : და ++.

ghci> 'a':"bc"

"abc"

ghci> "foo" ++ "bar"

"foobar"

[http://www.microsofttranslator.com/static/img/tooltip_logo.gif?156769](http://www.microsofttranslator.com/)http://www.microsofttranslator.com/static/img/tooltip_close.gif?156769

Original

Getting started with ghci, the interpreter