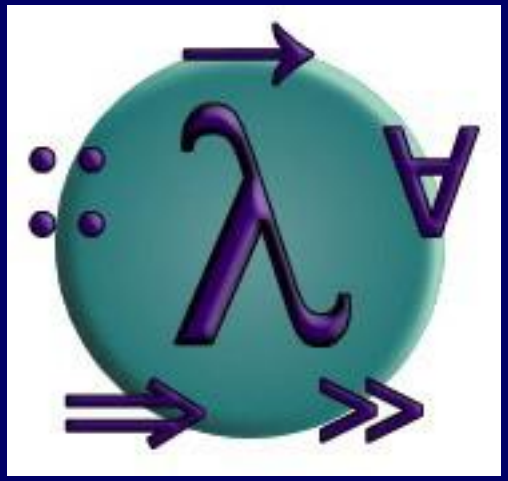
PROGRAMMING IN HASKELL
プログラミングHaskell



Chapter 2 - First Steps
はじめの一歩

愛知県立大学 情報科学部 計算機言語論(山本晋一郎・大久保弘崇、2011年)講義資料
オリジナルは http://www.cs.nott.ac.uk/~gmh/book.html を参照のこと

Starting Hugs

- UNIX では、シェルプロンプトから hugs (ghci) を起動する
- Windows では、スタートメニュの Haskell Platformから、GHCi か WinGHCi を起動する

```
% hugs

__   __ __  __  ____   ___                ________________________________________________
||   || ||  || ||  || ||__                Hugs 98: Based on the Haskell 98 standard
||___|| ||__|| ||__||  __||               Copyright (c) 1994-2005
||---||         ___||                     World Wide Web: http://haskell.org/hugs
||   ||                                   Bugs: http://hackage.haskell.org/trac/hugs
||   || Version: September 2006 _____________________________________________

Haskell 98 mode: Restart with command line option -98 to enable extensions

Type :? for help
Hugs>
```

```
% ghci
GHCi, version 6.12.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude>
```

Hugs のプロンプト > は Hugs システムが式を評価できることを表す

For example:

```
> 2+3*4
14

> (2+3)*4
20

> sqrt (3^2 + 4^2)
5.0
```

標準プレリュード

ライブラリファイル Prelude.hs は膨大な数の標準装備の関数を提供する。+ や * のような数学関数に加えて、リストに対する有用な関数も多数提供されている。

- 空でないリストの先頭要素を取り出す:

```
> head [1,2,3,4,5]
1
```

空リストの head は?

```
> head []
*** Exception: Prelude.head: empty list
```

- 空でないリストの先頭要素を除いたリスト:

```
> tail [1,2,3,4,5]
[2,3,4,5]
```

- 非空リストの(0から) n 番目の要素を返す演算子:

```
> [1,2,3,4,5] !! 2
3
```

- リストの先頭 n 個の要素から成る部分リスト:

```
> take 3 [1,2,3,4,5]
[1,2,3]
```

- リストから先頭 n 個の要素を除いた部分リスト:

```
> drop 3 [1,2,3,4,5]
[4,5]
```

- リストの長さ:

```
> length [1,2,3,4,5]
5
```

- 数値リストの要素の和:

```
> sum [1,2,3,4,5]
15
```

- 数値リストの要素の積:

```
> product [1,2,3,4,5]
120
```

- リストの結合:

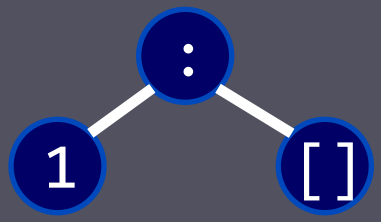
```
> [1,2,3] ++ [4,5]
[1,2,3,4,5]
```

- リストの反転:

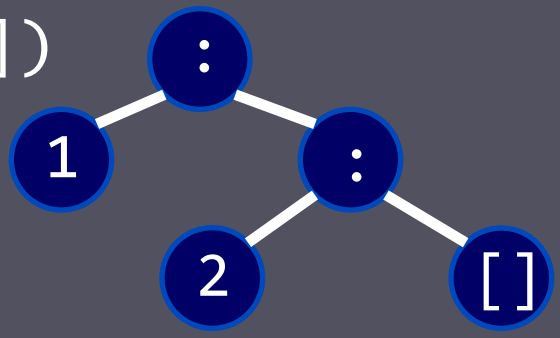
```
> reverse [1,2,3,4,5]
[5,4,3,2,1]
```

リストの復習(Cons 演算子)

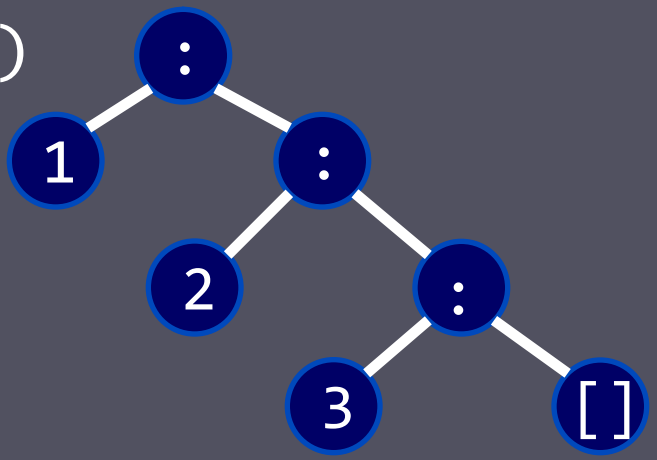
- `[1] = 1:[]`



- `[1,2] = 1:(2:[])`



- `[1,2,3] = 1:(2:(3:[]))`



関数適用(Function Application)

数学では、関数適用を括弧で、項の連続で乗算を表現

```
f(a,b) + c d
```

関数 f を a と b に適用し、
その結果に c と d の積を足す

Haskell では、関数適用は空白で、乗算は * で表現

```
f a b + c * d
```

前のシートと同じ、ただし Haskell 記法

さらに、関数適用は全ての演算子より結合順位が高い

```
f a + b
```

計算順は (f a) + b であって f (a + b) ではない

Examples

Mathematics	Haskell
`f(x)`	`f x`
`f(x,y)`	`f x y`
`f(g(x))`	`f (g x)`
`f(x,g(y))`	`f x (g y)`
`f(x)g(y)`	`f x * g y`

Haskell スクリプト

- 標準 Prelude で提供される関数と同様に、自分で新しい関数を定義できる
- 関数は定義の並びとしてスクリプトと呼ぶテキストファイルに記述する
- 慣習として、Haskell ソースの拡張子は .hs とする
  - 強制ではないが、標準に従う方が区別に便利

My First Script

Haskell プログラムを書くときには、2 つのウィンドウを開いておくと便利。一つでプログラムを編集するエディタを動かし、もう一つで Hugs を動かす。

エディタを起動し、以下の2 つの関数定義を入力し、test.hs という名前で保存する:

```
double x     = x + x

quadruple x = double (double x)
```

エディタを開いたまま、別のウィンドウで以下のコマンドにより Hugs を起動する:

```
% hugs test.hs
```

Prelude.hs と test.hs の両方が読込まれているので、どちらで定義された関数も使える:

```
> quadruple 10
40

> take (double 2) [1,2,3,4,5,6]
[1,2,3,4]
```

Hugs を開いたままエディタに戻り、以下の定義を追加して test.hs を上書き保存する:

```
factorial n = product [1..n]

average ns  = sum ns `div` length ns
```

Note:

- div は逆クオート([`] 日本語キーボードでは Shift + @)で囲まれている
- x `f` y は f x y の糖衣構文(syntactic sugar)

Hugs はソースファイルが更新されても自動的に検知しないので、新しい定義を使うために再読込み命令を実行する:

```
> :reload
Reading file "test.hs"

> factorial 10
3628800

> average [1,2,3,4,5]
3
```

命名規則(Naming Requirements)

- 関数と引数の名前は小文字で始める
  例:

```
myFun    fun1    arg_2    x'
```

- 慣習として、リスト引数の名前は最後に s をつける
  例:

```
xs    ns    nss
```

xs は任意のリスト、
ns は数値のリスト、
nss は数値リストのリスト

レイアウト規則

定義を並べるとき、(同じレベルの)全ての定義は同じカラムにインデントする:

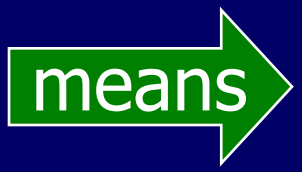
```
a = 10
b = 20
c = 30
```

```
a = 10
 b = 20
c = 30
```

```
  a = 10
b = 20
  c = 30
```

レイアウト規則により、定義のグループ化を表すための明示的な文法が不必要になる

```
a = b + c
    where
      b = 1
      c = 2
d = a * 2
```




```
a = b + c
    where
      {b = 1;
       c = 2}
d = a * 2
```

暗黙のグループ化

明示的なグループ化

役に立つHugs (ghci)コマンド

Command	Meaning
:load *name*	load script *name*
:reload	reload current script
:edit *name*	edit script *name*
:edit	edit current script
:type *expr*	show type of *expr*
:?	show all commands
:quit	quit Hugs

まとめ(2章)

- 標準ライブラリ Prelude.hs
  - 数やリストなどの基本的なデータ構造を定義
- 関数適用
  - 数学: `f(a,b) + c d`
  - Haskell: `f a b + c * d`
    - 関数適用は空白(引数の括弧は不要)
    - 全ての演算子より高い結合順位(関数適用は重要なので)
- Haskell スクリプトの拡張子は ".hs"
- 命名規則
  - 関数と引数は小文字で始まり、リスト引数は s で終る
- レイアウト規則
  - (同じレベルの)全ての定義は同じカラムにインデントする