

Comp 205: Comparative Programming Languages

Functional Programming Languages:

Haskell

Lecture notes, exercises, etc., can be found at:
www.csc.liv.ac.uk/~grant/Teaching/COMP205/

An Introduction to Haskell

- The interpreter
- Types
- Expressions
- Functions
- Evaluation

Starting Hugs

```
$ which hugs
/usr/bin/hugs
$ hugs
...
Reading file ".../Prelude.hs"

Hugs session for:
/usr/share/hugs/lib/Prelude.hs
Type :? for help
Prelude>
```

Interpreting What?

The interpreter does two things:

- Evaluate expressions
- Evaluate commands, e.g.
 - `:quit quit`
 - `:load <file>` load a file
 - `:r` redo the last load command

Interpreting Expressions

```
Prelude>
```

Interpreting Expressions

```
Prelude> 3.123456789012345
3.12346
Prelude>
```

Interpreting Expressions

```
Prelude> 3.123456789012345
3.12346
Prelude> 3.1234567 < 3.1234569
True
Prelude>
```

Interpreting Expressions

```
Prelude> 3.123456789012345
3.12346
Prelude> 3.1234567 < 3.1234569
True
Prelude> 3 == sqrt 9.0000000000000001
True
Prelude>
```

Interpreting Expressions

```
Prelude> 3.123456789012345
3.12346
Prelude> 3.1234567 < 3.1234569
True
Prelude> 3 == sqrt 9.0000000000000001
True
Prelude> 3 / 1.0
3.0
Prelude>
```

Interpreting Expressions

```
Prelude> 3.123456789012345
3.12346
Prelude> 3.1234567 < 3.1234569
True
Prelude> 3 == sqrt 9.0000000000000001
True
Prelude> 3 / 1.0
3.0
Prelude> 4 / 2
2.0
Prelude>
```

Types

Integer	Unbounded integer numbers
Int	32-bit integer numbers
Rational	Unbounded rational numbers
Float, Double	Single- and double-precision floating point numbers
Bool	Boolean values
Char	Characters, e.g., 'a'

Expressions

In Haskell, expressions are built from:

- constants
- operators
- function applications
- (brackets)

Expressions

Constants are literal denotations, e.g.:

- -3.459
- True
- 'a'
- "hello"
- [1,2,4,9]

Function application uses prefix notation:

- even 47
- twice (-2)

Operators

.	function composition
*, /, ^	multiply, divide, power
*, -	addition, subtraction
:, ++	add to list, concatenate
==, /=, <=, ...	equals, unequal, comparison
&&	logical and
	logical or

Definitions

Programmers can define constants and functions

Definitions must be contained in a file (e.g., arithmetic.hs)

A constant simply introduces an alias for a value:

```
-- approximation to mathematical pi
pie = 3.14159
```

Functions

Function definitions have the form:

<Name> <Var> = <Exp>

```
-- compute circumference of a circle
-- given its radius

circumference r = 2 * pie * r

-- area of a circle
area rad = pie * rad^2
```

The Offside Rule

Indentation determines where a definition ends:

```
circumference r =
  2 * pie * r

area r
  = pie * r * r

bad x = area x
+ circumference x      -- offside!
```

Declaring Types

Function definitions can include type declarations

```
circumference :: Float -> Float
circumference r = 2 * pie * r

area :: Double -> Double
area r = pie * r ^ 2
```

Hugging Files

```
Prelude> :l arithmetic
```

```
Main> area 1  
3.14159  
Main>
```

Pattern-Matching #1

Functions can be defined using more than one equation:

```
-- factorial of a number  
  
fact :: Integer -> Integer  
  
fact 0 = 1  
fact n = fact (n-1) * n
```

Summary

Key topics:

- Expressions
- Types
- function definitions
- pattern-matching

Next: More Haskell