CS1JC3-Sept13-15

Welcome to CS 1JC3 - Intro To Computational Thinking

Starting ghci

- In this course, we will be using an open source haskell interpreter known as ghci
- To run ghci, you must first access your systems command line interface or terminal, or you can open winghci, or sublime repl
- Once you have opened your command line, type ghci and hit enter. You are now running ghci!

Try This

The command line should now display Prelude> followed by the cursor. GHCi will evaluate any valid expression in haskell, and can be used as a powerful calculator. Try typing in the following.
(Do not type Prelude>)

```
Prelude> 2+3*4
Prelude> (2+3)*4
Prelude> sqrt (3^2 + 4^2)
```

Lists

- When programming, we often wish to group together values. In haskell, one way of accomplishing this is through lists.
- Lists are created by putting values inside square brackets, seperated by commas
- For example, [1,5,3,2] is a list of numbers
- `[] is an example of an empty list`

Function Application

- In **Haskell**, function application is denated using space, ex:

```
function var1 var2
```

- Moreover function application is assumed to have higher priority than all other operators, ex:

```
sin 0+2 = (sin 0) + 2
```

Examples

Mathmatics

f(x)
f(x,y)
f(g(x))
f(x,g(y))
f(x)g(y)

Examples

Mathmatics	**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

The Standard Prelude

- When you run ghci, it automatically loads **The Standard Prelude**, a module containing a large number of standard functions.
- In addition to the familiar numeric functions such as $+$ and *, the library also provides many useful functions on **lists**, which will be covered in more detail later in the course.

Some Useful Functions

- Select the first element of a list

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

- Remove the first element of a list

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

- Select the nth element of a list, ie [1,2,3] !! n

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

Note: the first element is index 0
```

Some Useful Functions

- Select the first element of a list

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

- Remove the first element of a list

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

- Select the nth element of a list, ie [1,2,3] !! n

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

Some Useful Functions

- Remove the first n elements of a list

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

- Calculate the length of a list

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

- Calculate the sum of a list of numbers

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

Some Useful Functions

- Remove the first n elements of a list

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

- Calculate the length of a list

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

- Calculate the sum of a list of numbers

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

Some Useful Functions

- Calculate the product of a list of numbers

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

- Append two lists

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

- Reverse a list

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

Some Useful Functions

- Calculate the product of a list of numbers

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

- Append two lists

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

- Reverse a list

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

Creating Your Own Functions

- As well as the functions in the standard prelude, you can also define your own functions
- New functions are defined within a text file comprising a sequence of definitions
- By convention, Haskell files usually have a **.hs** suffix on their filename.

Creating Your Own Functions

- Start an editor, type in the following two functions, and save the script as **test.hs**
  Note: it doesn't matter how much spacing you use, as long as there is a space between the function and its arguments

```
double x      = x + x
quadruple x   = double (double x)
```

- From ghci, browse to the directory your file is by typing **:cd directory**, or click the top left folder in winghci

Loading Your Functions

- Load your functions by executing **:load test.hs**, or if your in winghci simply browse to the file and double click
- Now both the Prelude and test.hs are loaded, and functions from both can be used
- Try executing the following

```
*Main> quadruple 10
*Main> take (double 2) [1,2,3,4,5,6]
```

* take x (this takes the first x number of arguments in a list *

More Functions

- Leaving ghci open, return to the editor, add the following two functions and resave

```
factorial n = product [1 .. n]
average ns  = sum ns `div` length ns
```

- Note : div is enclosed in **back** quotes, not forward
  x `f` y is just **syntactic sugar** for f x y

Reload

- GHCi does not automatically detect the file has been changed, to do so type **:reload**, or hit the top green button in winghci
- Try executing some of our new functions

```
*Main> factorial 10

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

Reload

- GHCi does not automatically detect the file has been changed, to do so type **:reload**, or hit the top green button in winghci
- Try executing some of our new functions

```
*Main> factorial 10
       3628800

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

Naming Requirements

- Function and argument (variable) names must begin with a lower-case letter. For example:

```
myFun      fun1      arg_2     x'
```

* Naming must follow: camelCaseConvention

- By convention, lists usually have an **s** suffix on their name. For example :

```
xs    ns    nss
```

The Layout Rule

In a sequence of definitions, each definition must begin in precisely the same column

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

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

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

Use Spaces, NOT Tabs!!!

The Layout Rule

In a sequence of definitions, each definition must begin in precisely the same column

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

Right

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

Wrong

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

Wrong

Implicit vs Explicit

The layout rule avoids the need for explicit syntax to indicate the grouping of definitions

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

Implicit Grouping

In other words, we don't need shit like curly braces, because the spaces automatically tells the interpreter, what line is a part of what function

Implicit vs Explicit

The layout rule avoids the need for explicit syntax to indicate the grouping of definitions

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

Implicit Grouping

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

Explicit Grouping

But we can use braces

Some Useful Commands

If your using the terminal, here are some useful commands you can use inside ghci

Command	Meaning
:load **Name of file**	Loads specified file
:reload —> :r	Reloads current file
:edit **Name of file**	Edits specified file
:type **expr**	Displays type of expr
:quit —> :q	Exits ghci

Exercise 1

Fix the syntax errors

```
N = a 'div' length xs
    where
      a = 10
    xs = [1,2,3,4,5]
```

Fix the syntax errors

```
N = a ’div’ length xs
    where
      a = 10
    xs = [1,2,3,4,5]
```

Functions must start with a lowercase

Must use (`), and NOT (')

Solution

```
n = a ‘div‘ length xs
    where
      a  = 10
      xs = [1,2,3,4,5]
```

Must be indented

Exercise 2

Fix the syntax errors

```
f(x,y) = let
     z1 = x*x
      z2 = y*y
     z3 = z1 + z2
   in sqrt(z3)
```

Exercise 2

Fix the syntax errors

```
f(x,y) = let
     z1 = x*x
      z2 = y*y
     z3 = z1 + z2
   in sqrt(z3)
```

Not in line with other statements

Around brackets around ‘z3’ is not nescessary

Solution

```
f x y = let
     z1 = x*x
     z2 = y*y
     z3 = z1 + z2
   in sqrt z3
```

or

```
f (x,y) = let
     z1 = x*x
     z2 = y*y
     z3 = z1 + z2
   in sqrt z3
```

Note: the left solution uses an effect called **currying**, more on this later

Exercise 3

Show how the Prelude function **last** can be defined using other Prelude functions introduced in these slides
NOTE: (Call it **lastC** since **last** is already defined)

Note: The "last" function, returns the last item in a list

Exercise 3

Show how the Prelude function **last** can be defined using other Prelude functions introduced in these slides
NOTE: (Call it **lastC** since **last** is already defined)

Solution 1

```
lastC xs = head (reverse xs)
```

Exercise 3

Show how the Prelude function **last** can be defined using other Prelude functions introduced in these slides
NOTE: (Call it **lastC** since **last** is already defined)

Solution 1

```
lastC xs = head (reverse xs)
```

Solution 2

```
lastC xs = xs !! (length xs - 1)
```

Exercise 4

Now show how the Prelude function **init** can be defined in two different ways
NOTE: (Call it **initC** since **init** is already defined)

Note: The “init” function, drops the last item in a list

Exercise 4

Now show how the Prelude function **init** can be defined in two different ways

NOTE: (Call it **initC** since **init** is already defined)

Solution 1

```
initC xs = take (length xs - 1) xs
```

Exercise 4

Now show how the Prelude function **init** can be defined in two different ways

NOTE: (Call it **initC** since **init** is already defined)

Solution 1

```
initC xs = take (length xs - 1) xs
```

Solution 2

```
initC xs = reverse (tail (reverse xs))
```

Exercise 5

Define the index function **!!** using Prelude functions in these slides.
Note: (Call it **!!!** since **!!** is already defined)

[4, 5, 6, 7, 8] !! 3

^ This returns the 3rd element in the list.

Note: Counting starts at 0

Exercise 5

Define the index function **!!** using Prelude functions in these slides.
Note: (Call it **!!!** since **!!** is already defined)

Solution 1

```
xs !!! n = head (drop n xs)
```

Define the index function **!!** using Prelude functions in these slides.
Note: (Call it **!!!** since **!!** is already defined)

Solution 1

```
xs !!! n = head (drop n xs)
```

Solution 2

```
(!!!) xs n = last (take (n+1) xs)
```

Note++: One of these definitions doesn't crash when **n** exceeds **length xs**, which one? Why? Is this a feature or a bug?

Exercise 6

Create two functions, **firstHalf** and **lastHalf**, that well... return the first half and last half of a list respectively. Define the first half function first, and use it to define the last half function

Exercise 6

Create two functions, **firstHalf** and **lastHalf**, that well... return the first half and last half of a list respectively. Define the first half function first, and use it to define the last half function

Solution 1

```
firstHalf xs = take (length xs `div` 2) xs
```

Exercise 6

Create two functions, **firstHalf** and **lastHalf**, that well... return the first half and last half of a list respectively. Define the first half function first, and use it to define the last half function

Solution 1

```
firstHalf xs = take (length xs `div` 2) xs
```

Solution 2

```
lastHalf xs = reverse (firstHalf (reverse xs))
```

Exercise 7

Using Prelude functions introduced in these slides, create a function **inners** that removes the first and last element of a list (leaving just the inner part of the list)

Exercise 7

Using Prelude functions introduced in these slides, create a function **inners** that removes the first and last element of a list (leaving just the inner part of the list)

Solution 1

```
inners xs = reverse (tail (reverse (tail xs)))
```

Exercise 7

Using Prelude functions introduced in these slides, create a function **inners** that removes the first and last element of a list (leaving just the inner part of the list)

Solution 1

```
inners xs = reverse (tail (reverse (tail xs)))
```

Solution 2

```
inners xs = take (length xs - 2) (drop 1 xs)
```

Exericise 8

Implement a function for computing the **Euclidean distance** between two points, (x1,y1) and (x2,y2). In case you forget:

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Exericise 8

Implement a function for computing the **Euclidean distance** between two points, (x1,y1) and (x2,y2). In case you forget:

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Solution

```
dist (x1,y1) (x2,y2) = let
        xd = x2 - x1
        yd = y2 - y1
    in sqrt (xd^2 + yd^2)
```