Project Euler

Dr. Mattox Beckman

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF COMPUTER SCIENCE

Our Environment

- ► Create a new directory for this project.
- ► Using an editor, create a file euler.hs. Add the following two lines to it:

```
1 inc :: Integer -> Integer
2 inc x = x + 1
```

- ► Now type stack repl euler.hs.

Expected Output

```
% stack repl euler.hs

[1 of 1] Compiling Main             ( /home/mattox/euler/euler.hs, interpre
Ok, one module loaded.
Loaded GHCi configuration from /run/user/1000/ghci4772/ghci-script
*Main> :t inc
inc :: Integer -> Integer
*Main> inc 10
11
*Main> :r
Ok, one module loaded.
*Main>
Leaving GHCi.
```

Problem 1 – Multiples of 3 or 5

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6, and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.

Some Arithmetic

```
1 *Main> mod 10 3
2 1
3 *Main> 10 `mod` 3
4 1
5 *Main> 10 `mod` 3 == 0
6 False
7 *Main> 10 `mod` 3 == 0 || 10 `mod` 5 == 0
8 True
9 *Main> mod3or5 x = x `mod` 3 == 0 || x `mod` 5 == 0
10 *Main>
```

Add the following line to your euler.hs.

```
1 mod3or5 x = x `mod` 3 == 0 || x `mod` 5 == 0
```

A Type Constraint

- HASKELL will *infer* the types of things if you don't specify them!
- Let's see what it thinks of our new function ...

```
1 *Main> :t mod3or5
2 mod3or5 :: Integral a => a -> Bool
```

A Type Constraint

- ▶ HASKELL will *infer* the types of things if you don't specify them!
- ▶ Let's see what it thinks of our new function …

```
1 *Main> :t mod3or5
2 mod3or5 :: Integral a => a -> Bool
```

- ▶ "The input can be any type that is an Integral, and the output is a Bool."

List Operations

```
1 *Main> [3,5,7,9]
2 [3,5,7,9]
3 *Main> map inc [3,5,7,9]
4 [4,6,8,10]
5 *Main> map mod3or5 [3,5,7,9]
6 [True,True,False,True]
7 *Main> filter mod3or5 [3,5,7,9]
8 [3,5,9]
9 *Main> sum (filter mod3or5 [3,5,7,9])
10 17
```

Put the following definition into your euler.hs:

```
1 sumMods xx = sum (filter mod3or5 xx)
```

So how do we get a list from 1 to 999?

Big Lists

```
1 *Main> [1..20]
2 [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
3 *Main> [1,3..20]
4 [1,3,5,7,9,11,13,15,17,19]
5 *Main> [1,5..20]
6 [1,5,9,13,17]
7 *Main> [1..999]  -- Go ahead and try it!
```

So now add this line to your euler.hs:

```
1 euler1 = sumMods [1..999]
```

Sample Run

```
1 *Main> :r
2 [1 of 1] Compiling Main              ( /home/mattox/euler/euler.hs, interp
3 Ok, one module loaded.
4 *Main> euler1
5 233168
```

Final Result

```
1 inc :: Integer -> Integer
2 inc x = x + 1
3
4   -- Euler Problem 1
5
6 mod3or5 x = x `mod` 3 == 0 || x `mod` 5 == 0
7 sumMods xx = sum (filter mod3or5 xx)
8 euler1 = sumMods [1..999]
```

We can clean this up a little …

Cleaner Version

```
1 inc :: Integer -> Integer
2 inc x = x + 1
3
4    -- Euler Problem 1
5
6 euler1 = sumMods [1..999]
7   where mod3or5 x = x `mod` 3 == 0 || x `mod` 5 == 0
8         sumMods xx = sum (filter mod3or5 xx)
```

- ▶ The where keyword introduces local definitions.
- ▶ Indentation determines the scope of definitions.
- ▶ Be sure your editor never uses tabs!!