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, interpr
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

```
*Main> mod 10 3
1
*Main> 10 `mod` 3
1
*Main> 10 `mod` 3 == 0
False
*Main> 10 `mod` 3 == 0 || 10 `mod` 5 == 0
True
*Main> mod3or5 x = x `mod` 3 == 0 || x `mod` 5 == 0
*Main>
```

Add the following line to your `euler.hs`.

```
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 …

```
*Main> :t mod3or5
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 …

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

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

List Operations

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

Put the following definition into your `euler.hs`:

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

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

Big Lists

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

So now add this line to your `euler.hs`:

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


Sample Run

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


Final Result

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

  -- Euler Problem 1

mod3or5 x = x `mod` 3 == 0 || x `mod` 5 == 0
sumMods xx = sum (filter mod3or5 xx)
euler1 = sumMods [1..999]
```

We can clean this up a little …

Cleaner Version

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

  -- Euler Problem 1

euler1 = sumMods [1..999]
  where mod3or5 x = x `mod` 3 == 0 || x `mod` 5 == 0
        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!!