Setting Up
 Problem 1
 Problem 3
 Problem 20
 Setting Up
 Problem 1
 Problem 3
 Problem 20

 oo
 oo<

Project Euler

Dr. Mattox Beckman

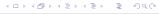
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF COMPUTER SCIENCE

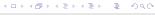
Our Environment

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

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

▶ Now type stack repl euler.hs.





Setting Up	Problem 1	Problem 3	Problem 20	Setting Up	Problem 1	Problem 3	Problem 20
0●	00000000	0000000	000	00	•0000000	0000000	000
							<i></i>

Expected Output

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.



 Setting Up
 Problem 1
 Problem 3
 Problem 20
 Setting Up
 Problem 1
 Problem 3
 Problem 20

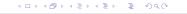
 oo
 oo<

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.
```

Problem 1

0000000



Problem 20

Setting Up

Problem 3

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
```



◆□▶◆御▶◆恵▶◆恵▶ 恵 め९♡

A Type Constraint

Setting Up

- ► 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
2mod3or5 :: 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
```

Problem 1

00000000

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?



Problem 1 Problem 3 Problem 1 Problem 3 Problem 20 Setting Up Problem 20 Setting Up 00000000 00000000

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 \text{ euler1} = \text{sumMods} [1..999]
```

Sample Run

```
1*Main> :r
2 [1 of 1] Compiling Main
                                       ( /home/mattox/euler/euler.hs, interp
30k, one module loaded.
4 *Main> euler1
5 233168
```



Problem 3 Problem 20

Setting Up

Problem 1 0000000 Problem 3

<ロト 4回 ト 4 重 ト 4 重 ト ■ 9 9 0 0 Problem 20

Final Result

Setting Up

```
inc :: Integer -> Integer
2 inc x = x + 1
  -- Euler Problem 1
7 sumMods xx = sum (filter mod3or5 xx)
8 euler1 = sumMods [1..999]
```

We can clean this up a little ...

Problem 1

00000000

Cleaner Version

```
inc :: Integer -> Integer
2 inc x = x + 1
  -- Euler Problem 1
6 euler1 = sumMods [1..999]
   where mod3or5 x = x \mod 3 == 0 \parallel 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!!

Problem 1 Problem 3 Problem 1 Problem 3 Problem 20 Setting Up Problem 20 Setting Up •000000 000000

Euler Problem 3 – Prime Factors

The prime factors of 13195 are 5, 7, 13, and 29. What is the largest prime factor of the number 600851475143?

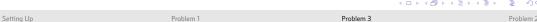
Sectioning

```
1*Main> plus a b = a + b
2 *Main> :t plus
3 plus :: Num a => a -> a -> a
4 *Main> plus 10 20
5 30
6 *Main> :t (plus 1)
7 (plus 1) :: Num a => a -> a
8 *Main> addTwo = plus 2
9 *Main> addTwo 10
10 12
```

► You can also say things like (+1) to get a partially applied operator.



4□ > 4団 > 4 豆 > 4 豆 > 9 Q @



0000000

Problem 20

Setting Up

Problem 1

Problem 3 0000000

<ロト 4 回 ト 4 亘 ト 4 亘 ト 9 Q ()

Problem 20

The Sieve

▶ We will make something like the Sieve of Eratosthenes.

```
*Main> notDivides a n = n `mod` a /= 0
2 *Main> notDivides 2 10
3 False
4 *Main> notDivides 3 10
5 True
6 *Main> filter (notDivides 3) [1..10]
7 [1,2,4,5,7,8,10]
```

Go ahead and add the definition of notDivides to your file.

Building Up Lists

- ▶ The operator : creates a list from an element and another list.
- ► HASKELL "a:b" is like JAVA/C++ "new Node(a,b)."
- ▶ The built-in function head will get you the first element of a list.

```
1*Main> 2 : filter (notDivides 2) [2..20]
2 [2,3,5,7,9,11,13,15,17,19]
3 *Main> 2 : filter (notDivides 2)
           (3 : filter (notDivides 3) [2..20])
5 [2,3,5,7,11,13,17,19]
6 *Main> 2 : filter (notDivides 2)
          (3 : filter (notDivides 3)
             (5 : filter (notDivides 5) [2..20]))
9 [2,3,5,7,11,13,17,19]
```

We need a recursive solution for this!



Problem 1 Problem 3 Problem 1 Problem 3 Problem 20 Setting Up Problem 20 Setting Up 0000000 00000•0

Making the Sieve

```
sieve (x:xs) = x : (sieve (filter (notDivides xs) xs))
3 primes = sieve [2..]
```

Sample Run

```
1 *Main> sieve [2..20]
2 [2,3,5,7,11,13,17,19,*** Exception: Prelude.head: empty list
3 *Main> take 20 (primes)
4 [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71]
5 *Main> take 20 $ primes
6 [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71]
```



Problem 3

000000

<ロト 4回 ト 4 重 ト 4 重 ト ■ 9 9 0 0 Problem 20 Problem 1 Problem 3 Setting Up Problem 20

Factors

Setting Up

▶ Now to get the factors ...

```
1 factors n = aux n primes
   where aux 1
         aux n (p:ps) = case divMod n p of
                           (n', 0) \rightarrow p : aux n' ps
                           (_ , _) -> aux n ps
7 maxFactor n = foldr1 max $ factors n
9 euler3 = maxFactor 600851475143
1*Main> foldr1 (+) [2,3,4,5]
2 14
3 *Main> euler3
46857
                                                      →□▶→□▶→□▶→□▶ □ りへで
```

Problem 1

Problem 20 - Factorial Digit Sum

```
n! means n \times (n-1) \times \cdots \times 3 \times 2 \times 1.
```

For example, $10! = 10 \times 9 \times \cdots \times 3 \times 2 \times 1 = 3628800$, and the sum of the digits in the number 10! is 3 + 6 + 2 + 8 + 8 + 0 + 0 = 27.

Find the sum of the digits in the number 100!



 Setting Up
 Problem 1
 Problem 3
 Problem 20
 Setting Up
 Problem 1
 Problem 3
 Problem 20

 oo
 oo<

BigInts

- ► Most functional languages have "Big Integers," constrained only by your computer's memory.
- ► To get started, here's the definition for factorial:

```
1 fact 0 = 1
2 fanc n = n * fact (n-1)
```

▶ If we run this on 100 it actually works!

```
1*Main> fact 100
```

- 2933262154439441526816992388562667004907159682643816214685
- **3929638952175999932299156089414639761565182862536979208272**



Divide and Conquer!

- ▶ To get the least significant digit, just take a modulus!
- ► To divide by 10 without remainder, just use div.

```
1 sumDigits 0 = 0
2 sumDigits n = n `mod` 10 + sumDigits (n `div` 10)
3
4 euler20 = sumDigits $ fact 100

Now try ...
1 *Main> euler20
2 648
```

