COMP105 Lecture 26

Lazy Lists

Laziness and lists

Lists are never evaluated until they are needed

```
take 2 [1..100]
```

```
→ 1 : take 1 [2..100]
```

```
→ 1 : 2 : take 0 [3..100]
```

```
→ 1 : 2 : []
```

```
→ [1,2]
```

`take 2 [1..10]` is as efficient as `take 2 [1..1000]`

Infinite lists

Laziness allows for **infinite** lists

```
all_1s = 1 : all_1s
```

```
ghci> take 4 all_1s
[1,1,1,1]
```

`all_1s` is the same as `[1,1..]`

Infinite lists

Laziness allows us to do infinite **computations** on infinite lists

```
all_1s = 1 : all_1s

all_2s = zipWith (+) all_1s all_1s
```

```
ghci> take 4 all_2s
[2,2,2,2]
```

Infinite lists

```
numbers n = n : numbers (n+1)

ghci> take 10 (numbers 0)
[0,1,2,3,4,5,6,7,8,9]



even_numbers = filter even (numbers 0)

ghci> take 10 even_numbers
[0,2,4,6,8,10,12,14,16,18]
```

Infinite lists

```
sieve (x:xs) = x : filter (\y -> y `mod` x /= 0)
                                    (sieve xs)

primes = sieve [2..]
```

```
ghci> take 10 primes
[2,3,5,7,11,13,17,19,23,29]

ghci> primes !! 1000
7927
```

Recap: The evaluation operator

The $ operator evaluates a function

```
ghci> head $ [1,2,3,4]
1

ghci> tail $ [1,2,3,4]
[2,3,4]
```

Strict evaluation

The `$!` operator does **strict evaluation**

- For most code you won't notice the difference
- But it can change the error outputs

```
func a b = if a then b else 0
```

```
ghci> func False $ (error "error")
0

ghci> func False $! (error "error")
*** Exception: error
```