Thunks …

… and Infinite Lists …

… and why you need to stop thinking like a programmer …

… and start thinking like a mathematician

Agenda

How to generate infinite lists

How to use them creatively as problem-solving tools

Design a function called "repeat" that takes the number 5 as its only argument and returns the infinite list:

```
[5,5,5,5,5, ...]
```

Design a function called "nats" that takes no arguments and returns the infinite list of all natural numbers:

```
[0,1,2,3,4,5,6,7,8, ...]
```

Design a function called "facts" that takes no arguments and generates an infinite list of all factorials

Design a function called "fibs" that takes no arguments and generates an infinite list of all Fibonacci numbers

Design a function called "cycle" that takes the finite list:

```
[1,2,3]
```

as its only argument and returns the infinite list:

```
[1,2,3,1,2,3,1,2,3,1,2,3, ...]
```

Design a function called "iterate" that takes two arguments:

- **the function** `(2*)`
- **the number** `1`

and returns the infinite list:

```
[1,2,4,8,16,32,64,128,256,512, ...]
```

Design a function called "fizzbuzz" that solves the FizzBuzz problem using infinite lists

Design a function called "pascal" that generates Pascal's triangle in the form of an infinite list

Design a simplified game of Pacman using infinite lists

Design a function called "repeat" that takes the number 5 as its only argument and returns the infinite list:

```
[5,5,5,5,5, ...]
```

Design a function called "repeat" that takes the number 5 as its only argument and returns the infinite list:

```
[5,5,5,5,5, ...]
```

Solution:

```
repeat x = xs where xs = x : xs
```

Design a function called "nats" that takes no arguments and returns the infinite list of all natural numbers:

```
[0,1,2,3,4,5,6,7,8, ...]
```

Design a function called "nats" that takes no arguments and returns the infinite list of all natural numbers:

```
[0,1,2,3,4,5,6,7,8, ...]
```

Solution:

```
nats = 0 : map (1+) nats
```

`[0..]` **is the syntactic sugar for nats**

A brief explanation of thunks

```
nats = 0 : map (1+) nats

nats          = 0 : <thunk>
map (1+) nats = 1 : <thunk>

nats          = 0 : 1 : <thunk>
map (1+) nats = 1 : 2 : <thunk>

nats          = 0 : 1 : 2 : <thunk>
map (1+) nats = 1 : 2 : 3 : <thunk>
```

Design a function called "facts" that takes no arguments and generates an infinite list of all factorials

Design a function called "facts" that takes no arguments and generates an infinite list of all factorials

Solution:

```
facts = 1 : zipWith (*) facts (tail facts)
```

What does "zip" do?

```
Prelude> zip [4,5] [10,11]
[(4,10),(5,11)]
```

What does "zipWith" do?

```
Prelude> zipWith (+) [4,5] [10,11]
[14,16]
Prelude> zipWith (*) [4,5] [10,11]
[40,55]
```

Design a function called "fibs" that takes no arguments and generates an infinite list of all Fibonacci numbers

Design a function called "fibs" that takes no arguments and generates an infinite list of all Fibonacci numbers

Solution:

```
fibs = 1 : 1 : zipWith (+) fibs (tail fibs)
```

Design a function called "cycle" that takes the finite list:

```
[1,2,3]
```

as its only argument and returns the infinite list:

```
[1,2,3,1,2,3,1,2,3,1,2,3, ...]
```

Design a function called "cycle" that takes the finite list:

```
[1,2,3]
```

as its only argument and returns the infinite list:

```
[1,2,3,1,2,3,1,2,3,1,2,3, ...]
```

Solution:

```
cycle [] = undefined
cycle xs = ys where ys = xs ++ ys
```

Design a function called "iterate" that takes two arguments:
- the function `(2*)`
- the number `1`
and returns the infinite list:

```
[1,2,4,8,16,32,64,128,256,512, ...]
```

Solution:

```
iterate f x = x : iterate f (f x)
```

Expressed differently, "iterate f x" generates an infinite list which follows the pattern:

```
[x, f x, (f . f) x, (f . f . f) x, ...]
```

```
['1', '2', 'Fizz', '4', 'Buzz', 'Fizz', '7', '8', 'Fizz',
'Buzz', '11', 'Fizz', '13', '14', 'FizzBuzz', '16', '17',
'Fizz', '19', 'Buzz', 'Fizz', '22', '23', 'Fizz', 'Buzz',
'26', 'Fizz', '28', '29', 'FizzBuzz', '31', '32', 'Fizz',
'34', 'Buzz', 'Fizz', '37', '38', 'Fizz', 'Buzz', '41',
'Fizz', '43', '44', 'FizzBuzz', '46', '47', 'Fizz', '49',
'Buzz', 'Fizz', '52', '53', 'Fizz', 'Buzz', '56', 'Fizz',
'58', '59', 'FizzBuzz', '61', '62', 'Fizz', '64', 'Buzz',
'Fizz', '67', '68', 'Fizz', 'Buzz', '71', 'Fizz', '73',
'74', 'FizzBuzz', '76', '77', 'Fizz', '79', 'Buzz',
'Fizz', '82', '83', 'Fizz', 'Buzz', '86', 'Fizz', '88',
'89', 'FizzBuzz', '91', '92', 'Fizz', '94', 'Buzz',
'Fizz', '97', '98', 'Fizz', 'Buzz']
```

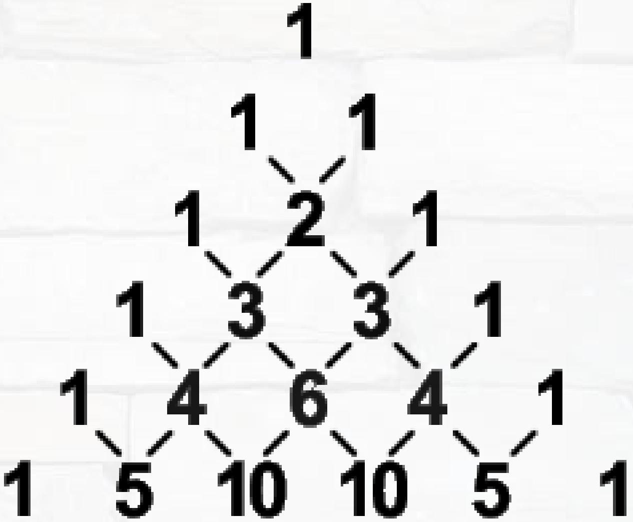
Design a function called "fizzbuzz" that solves the FizzBuzz problem using infinite lists. To speed things up, here's FizzBuzz in Python:

```
fizzbuzz = []
for num in range(1,101):
    msg = ''
    if num % 3 == 0:
        msg += 'Fizz'
    if num % 5 == 0:
        msg += 'Buzz'
    if not msg:
        msg += str(num)
    fizzbuzz.append(msg)
print fizzbuzz
```

Solution:

```
fizzbuzz_infinite_list =
  zipWith3 msg
           [1..]
           fizz_infinite_list
           buzz_infinite_list
  where
    fizz_infinite_list = cycle ["","","Fizz"]
    buzz_infinite_list = cycle ["","","","","Buzz"]
    msg e1 e2 e3 =
      if concat_e2_e3 == "" then show e1
                            else concat_e2_e3
      where
        concat_e2_e3 = e2 ++ e3

fizzbuzz = take 100 $ fizzbuzz_infinite_list
```



1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

Design a function called "pascal" that generates Pascal's triangle in the form of an infinite list

Design a function called "pascal" that generates Pascal's triangle in the form of an infinite list

Solution:

```
next row = zipWith (+) ([0] ++ row) (row ++ [0])
pascal = iterate next [1]
```

Design a simplified game of Pacman using infinite lists

```
▪   ▪   ▪   ▪   ▪
▪               ○
▪               ▪
▪               ▪
▪   P   ▪   ▪   ▪
○               ▪
▪               ▪
▪               ▪
▪   ☠   ▪   ▪   ○
```

Follow me on GitHub

github.com/dserban