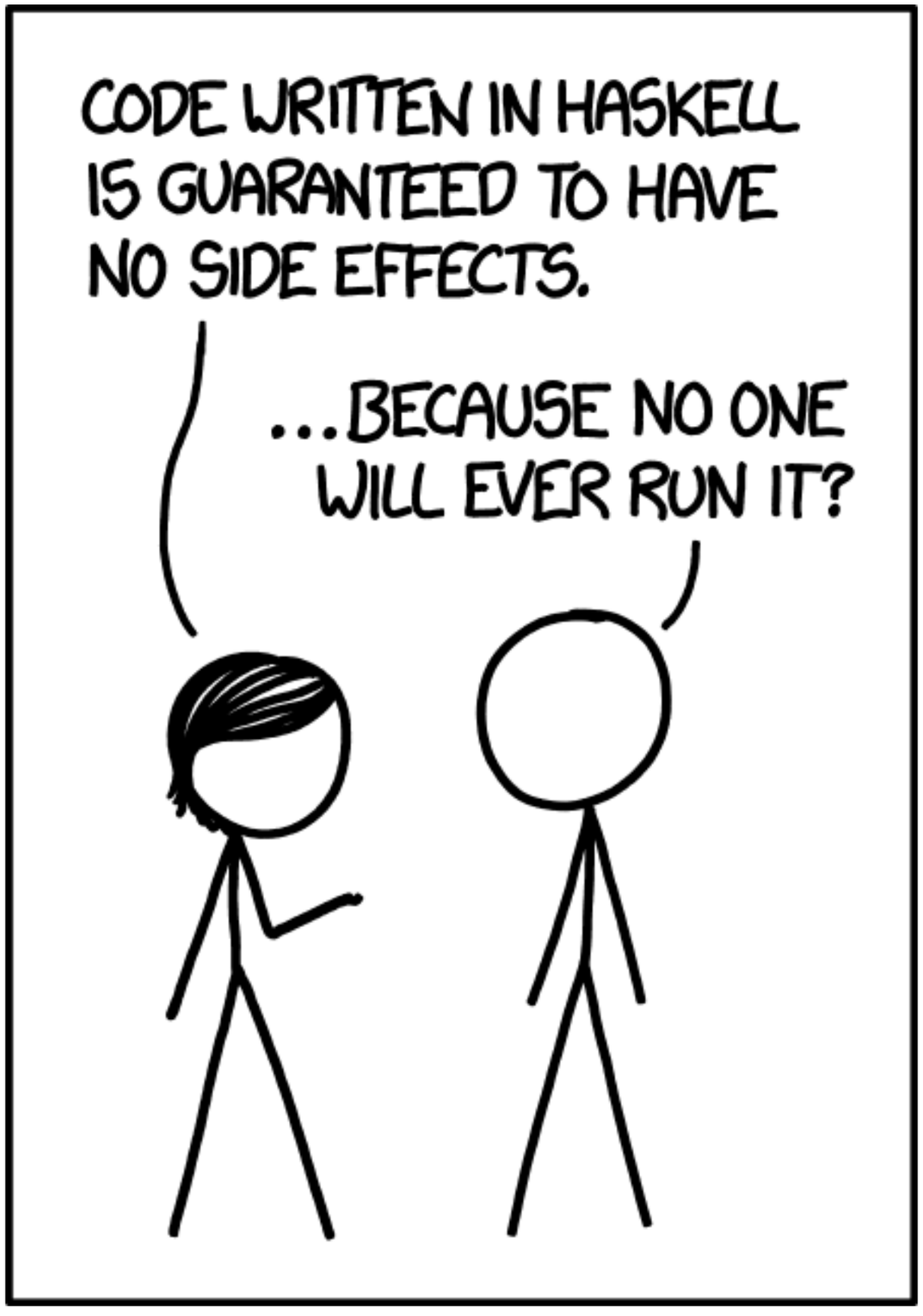
(and functional programming)

Useful	**C/C++** **C#** **Java** **JavaScript** **Python**	
Useless		**Haskell**
	Dangerous	**Safe**


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

Why Haskell?

- Strong influence on modern languages and patterns
- Teaches you to think differently about code
- Useful in its own right
- Fun

What is Haskell?

- Purely functional
- Lazily evaluated
- Strong type system

What is functional programming?

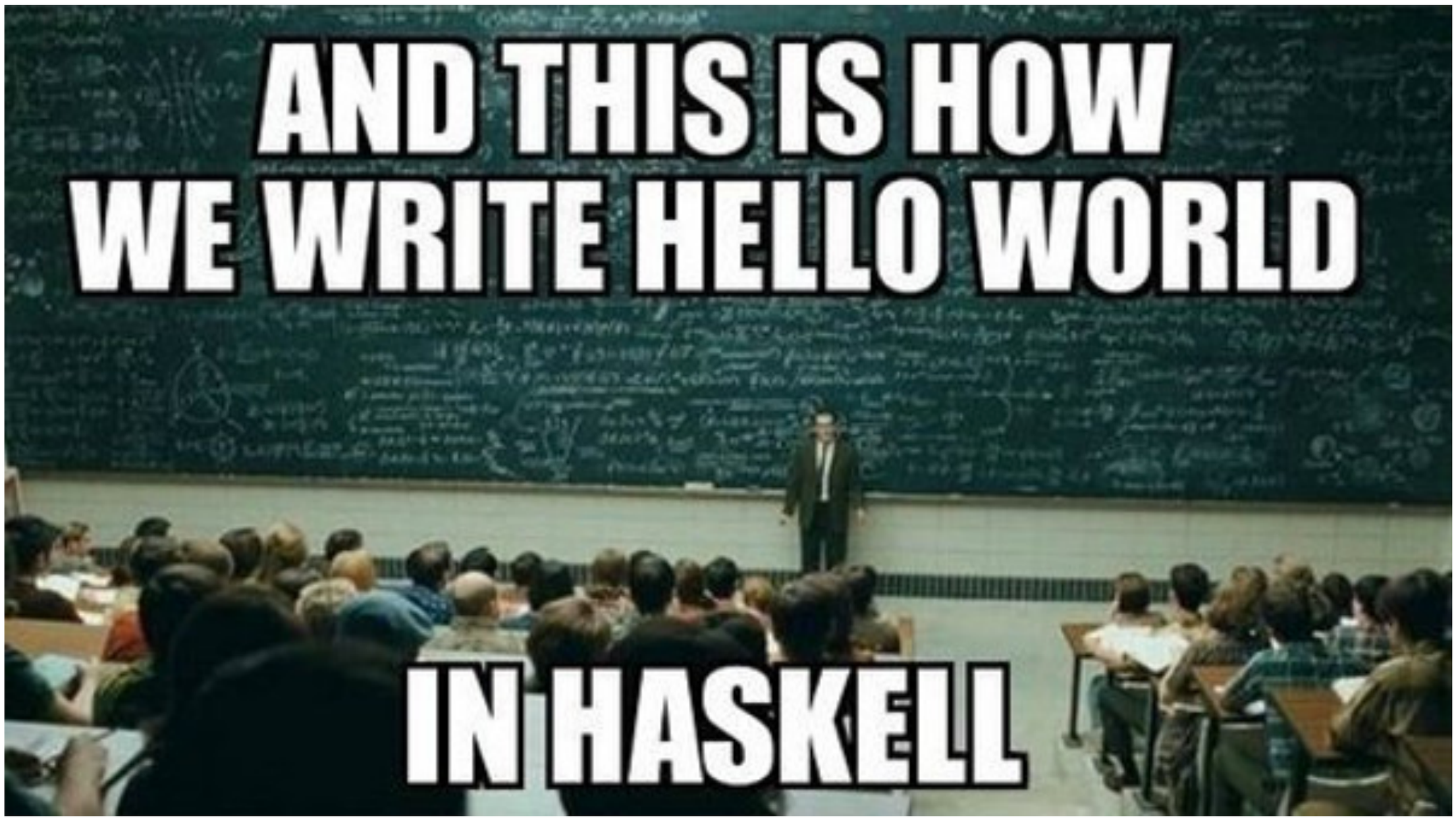
- Functions are first-class citizens (higher order functions)
- Functions are mathematically pure (no side effects)
- Code is modular and composed (concise and easy to reason about)
- Declarative vs imperative (what vs how)
- Data is immutable
- Recursion vs iteration

JavaScript Example

```
// Imperative
const function1 = (numbers, result) => {
  for (let i = 0; i < numbers.length; i++) {
    if (numbers[i] > 0) {
      result.push(numbers[i] * 2)
    }
  }
};

// Functional
const function2 = (numbers) =>
  numbers
    .filter(n => n > 0)
    .map(n => n * 2);
```

Demo



Hello World

```
main = putStrLn("Hello World!")
```

Functions

```
x = 1

increment n = n + 1

factorial 0 = 1
factorial n = n * factorial (n - 1)
```

Currying

```
// ES6
const add = x => y => x + y;
const add1 = add(1);
console.log(add1(2) == add(1)(2)); // true
```

```
// Old Way
function oldAdd(x) {
  return function (y) {
    return x + y;
  };
}
```

Currying

```haskell
-- Haskell
add :: Integer -> Integer -> Integer
add x y = x + y
-- add 1 2 == (add 1) 2

increment' = add 1
```

Composition

- Creating new functions by combining functions

```
// JavaScript
const incrementFactorial = n =>
  increment(factorial(n))
```

```
-- Haskell
incrementFactorial n = increment (factorial n)

incrementFactorial' = increment . factorial
```

Original Example

```javascript
// JavaScript
const exampleFunction = (numbers) =>
  numbers
    .filter(n => n > 0)
    .map(n => n * 2);
```

```haskell
-- Haskell
sampleFunction :: [Integer] -> [Integer]
sampleFunction = map (*2) . filter (>0)
```

Haskell Lists

- Linked lists vs random access arrays
- Optimized for recursion
- Lazily evaluated

```
list1 = [1, 2, 3, 4]
list2 = [1..4]
list3 = [2,4..10]
list4 = 0:list3
list5 = 1:2:3:4
```

Recursion vs Iteration

```
// JavaScript
const map = (transform, array) => {
  const result = [];
  for (const item of array) {
    result.push(transform(item));
  }
  return result;
};

-- Haskell
map' _ [] = []
map' f (item:rest) = (f item):(map' f rest)
```

Lazy Evaluation

```
import Math.NumberTheory.Primes.Sieve (primes)

evens = [0,2..]

sumOfEvens n = sum $ take n evens

sumOfFirst50Primes = sum $ take 50 primes

-- primes = 2:3:minus [5,7..] unionAll notPrimes
--     where notPrimes =
--           [[p*p, p*p+2*p..] | p <- tail primes]
```

Strong Typing

- Concise and elegant
- If it compiles it will work
- Together with pure functions guarantees behavior of code

CLI Example

- Reverse individual words in stdin
- https://github.com/jawang35/haskell-tech-talk/transform

```haskell
transform :: String -> String
transform = (++ "\n") . unwords . map reverse . words

main = interact transform
```

HTTP API Example

- API built with Servant
- https://github.com/jawang35/haskell-tech-talk/haskell-demo-api

Haskell at Surfline?

- Microservices in Haskell
- Data platform services
- Elm on the frontend

Resources

- What Haskell taught us when we were not looking!
- Fighting Spam with Haskell
- Learn You a Haskell
- Practical Haskell
- What I Wish I Knew When Learning Haskell
- Write Yourself a Scheme in 48 Hours
- An Opinionated Guide to Haskell in 2018