









- For developers
- For management
- For company

Developers

Compiler support



Advanced type system




```
data USD  
data EUR
```

```
newtype Currency a = Currency Double deriving Show
```

```
add :: Currency a → Currency a → Currency a  
add (Currency a) (Currency b) = Currency $ a + b
```

```
account :: Currency USD
```

```
account = Currency 5.0
```

```
transaction1 :: Currency USD
```

```
transaction1 = Currency 5.0
```

```
transaction2 :: Currency EUR
```

```
transaction2 = Currency 5.0
```

```
main = do
```

```
    print $ add account transaction1 - Currency 10.0
```

```
    print $ add account transaction2 - won't compile
```

```
$ ./currency_test.hs
```

```
currency_test.hs:23:25: error:
```


- Couldn't match type 'EUR' with 'USD'
Expected type: Currency USD
Actual type: Currency EUR
- In the second argument of 'add',
namely 'transaction2'
In the second argument of '(\$)',
namely 'add account transaction2'
In a stmt of a 'do' block:
print \$ add account transaction2

func a b = a + b

```
*Main> :t func  
func :: Num a => a -> a -> a  
*Main> █
```

Computations are more explicit

```
1 gtod = scd->tick_gtod + __gtod_offset;  
2 clock = gtod + delta;  
3 min_clock = wrap_max(gtod, old_clock);  
4 max_clock = wrap_max(old_clock, gtod + TICK_NSEC);
```



Explicit computation patterns

Developer> I have a database query here

Developer> it may fail, so it's kinda unsafe

Compiler> Don't worry, I'll check all the deps

Compiler> and let you know if some of them

Compiler> are not ok with that

```
calculation1 :: Int → Maybe Int  
calculation1 arg = Just (arg + 1)
```

```
calculation2 :: Int → Maybe Int  
calculation2 arg = Nothing
```

```
calculation3 :: Int → Maybe Int  
calculation3 arg = Just (arg + 2)
```

```
main = do
  print $ return 1 >>= calculation1 >>= calculation2
  print $ return 1 >>= calculation1 >>= calculation3
```


The same for state, IO,
sequential calculations etc.

Referential transparency a.k.a. refactoring without fear



Immutability

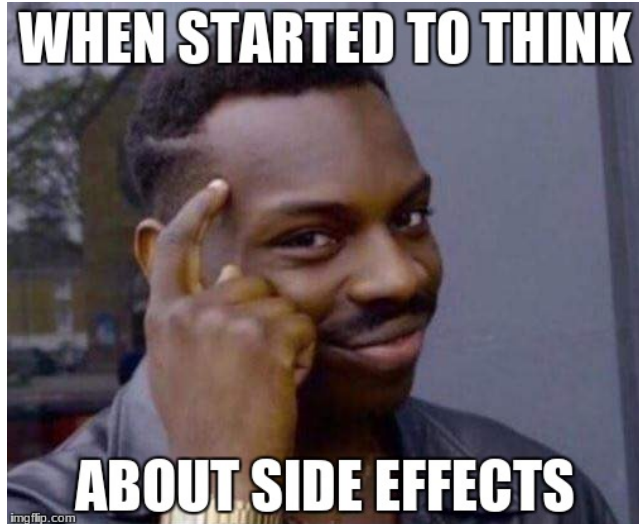
a.k.a. you can trust this function

```
def func(value=[ ]):  
    # whatever
```

someecards
user card



Explicit handling of side effects



Tooling support, great ecosystem

- stack
- ghc
- ghci
- editors and IDEs

Remarkable performance

Why is Haskell (GHC) so darn fast?



100



34

Haskell (with the `GHC` compiler) is a lot faster than you to low-level languages. (A favorite thing for Haskellers beat it, but that means you are using an inefficient C pr My question is, why?

Haskell is declarative and based on lambda calculus. I being based on turing machines, roughly. Indeed, Haskell order. Also, instead of dealing with machine data types

Do you need a parser?

Management

High quality code

Less bugs, easy to maintain and read.





youtube.com/watch?v=ybSBCVhVWs8

Awesome for agile development

Because of referential transparency
and how convenient is to create a DSL
it's easy to support and refactor large code base.

Heavy boost for skills improvement



You can attract a lot of talented people

You can attract a lot of talented people
For real

Company



wiki.haskell.org/Haskell_in_industry

Contribution is much more visible - not one of thousands nameless companies with the java technology stack, you have a ~~dragon~~ Haskell in production

YEAH, WELL, THAT'S JUST LIKE



**YOUR OPINION,
MAN**



Nerd




Geek

Questions?

 github.com/erthalion

 @erthalion

 9erthalion6 at gmail dot com