

10 modern programming concepts which your favourite programming language is missing¹



32th Chaos Communication Congress

December 29th, 2015

¹unless your favorite language is ...

Callback hell ☹

```
getData(function(a) {  
  getMoreData(a, function(b) {  
    getYetMoreData(b, function(c) {  
      getMoreFoo(c, function(d) {  
        ...  
      });  
    });  
  });  
});  
});
```

Callback hell ☹

```
getData(function(a) {  
  getMoreData(a, function(b) {  
    getYetMoreData(b, function(c) {  
      getMoreFoo(c, function(d) {  
        ...  
      });  
    });  
  });  
});  
});  
});
```

And this is even without error handling!

Overloaded semicolon 😊

```
do a <- getData  
   b <- getMoreData      a  
   c <- getYetMoreData b  
   d <- getMoreFoo       c
```

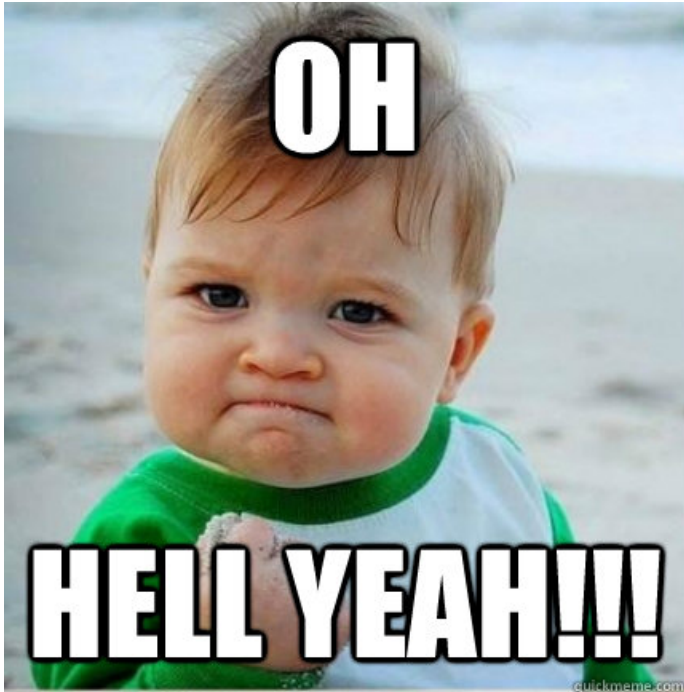
Simple & easy. You can pretend that you're using blocking I/O.

Overloaded semicolon 😊

```
do a <- getData  
   b <- getMoreData      a  
   c <- getYetMoreData  b  
   d <- getMoreFoo       c
```

Simple & easy. You can pretend that you're using blocking I/O.

NB: This is called “monads”. There are also monads for non-determinism, parsing, ...



QuickCheck

```
is(sqrt(4), 2, "sqrt(4) is working");  
is(sqrt(16), 4, "sqrt(16) is working");  
# ...
```

This does not scale! Better:

```
> quickCheck $ \x -> sqrt(x*x) == x  
*** Failed! Falsifiable (after 2 tests):  
-0.269864
```

```
> quickCheck $ \x -> sqrt(x*x) == abs x  
+++ OK, passed 100 tests.
```

Automatic test case generation.

Quiz time! Spot the error.

```
#include <stdlib.h>
int main(int argc, char *argv[]) {
    ...;
    user_input = ...;
    if(abs(user_input) > ...) {
        exit(1);
    }
    ...;
}
```


Quiz time! Spot the error.

```
#include <stdlib.h>
int main(int argc, char *argv[]) {
    ...;
    user_input = ...;
    if(abs(user_input) > ...) {
        exit(1);
    }
    ...;
}
```

Also: Million Dollar Mistake by Tony Hoare.

Solution: Option types.

Solution: Option types.

A value of type `Maybe Int` is

- 1 either `Nothing`
- 2 or a value of the form `Just x`, where `x` is an `Int`.

Type signature of `abs`: `Int -> Maybe Int`

Use option types when you cannot return a meaningful result and don't want to raise a proper exception.

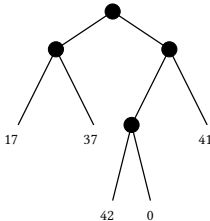
HEY GIRL. I JUST MET
YOU, AND THIS IS CRAZY.



BUT I WANT (JUST YOU).
SO CALL ME, MAYBE?

imgflip.com

Pattern matching



```
data Tree = Leaf Int | Fork Tree Tree
```

```
ex = Fork  
    (Fork (Leaf 17) (Leaf 37))  
    (Fork (Fork (Leaf 42) (Leaf 0)) (Leaf 41))
```

```
inorder :: Tree -> [Int]  
inorder (Leaf x)    = [x]  
inorder (Fork l r) = inorder l ++ inorder r
```

Typing

```
BufferedReader in =  
    new BufferedReader(  
        new InputStreamReader(System.in)  
    );
```



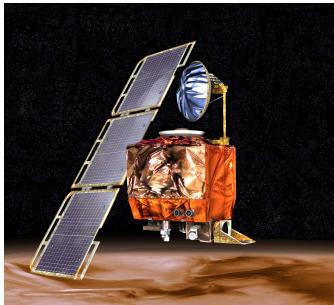
Types ☺

A good type system provides:

- inference: you don't have to type those types!
- safety: no `NullPointerException`
- “algebraic data types” and function types
- parametricity: generics on steroids
- higher-kinded types

Great for prototyping and refactoring!





Mars Climate Orbiter (1998)

Cost of the mission:
\$327 million

Failed due to a unit error
(newton-secs vs. pound-secs)



Units of Measure Types ☺

```
[<measure>] type N = (kg * m) / sec^2
```

```
fireThrusters (x:float<N * sec>) = ...
```

```
let duration = 2<sec>
```

```
let force = 1000<N>
```

```
fireThrusters (duration * force)
```

```
let diag (x:float<'u>) (y:float<'u>)  
    = sqrt (x*x + y*y) // Pythagoras
```



Time-traveling debugger

