10 modern programming concepts which your favourite programming language is missing[1]

Curry Club Augsburg

32th Chaos Communication Congress

December 29th, 2015

[1]unless your favorite language is Haskell, Idris or one of the ML-style languages (XXX)

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 :-)

```
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 :-)

```
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, . . .

Quiz time! Spot the error.

```c
#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.

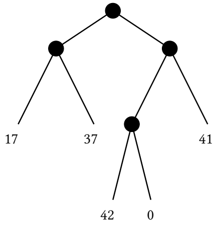
A value of type `Maybe Int` is

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.

Pattern matching



```haskell
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!