Programming languages - U3

Jan Dietrich - 10-100-436

1

The first definition:

```
func1 5 z = 33
```

Would only accept 5 as a first argument due to pattern matching and the second argument is unused.

The second definition:

```
func1 y z = y
```

mentions two element y z but only needs the first one for it's evaluation. The second one is a wildcard element that can be anything. Since haskell does lazy evaluation, it doesn't even evaluate the invalid expression `sqrt(-5)`

2

with pattern matching:

```
func 0 = -1
func n = n * 2
```

with guards:

```
func' n | n == 0 = -1
        | n >= 1 = n * 2
```

lambda expression:

note: My solution is with LambdaCase. To use it:

- Put -XLambdaCase on the command line, or
- Put {-# LANGUAGE LambdaCase #-} at the top of the file, or
- Run :set -XLambdaCase at the GHCi prompt

```
\case
        n | n == 0 -> -1
          | n >= 1 -> n * 2
```

can also be assigned to use:

```
func'' = \case
        n | n == 0 -> -1
          | n >= 1 -> n * 2
```

3 sum of list

There is a built in `sum` function that does exactly this. But anyway I did my own implementation with

```
sum' [ ] = 0
sum' (x:xs) = x + sum' xs
```

4 catalan

```
fak 0 = 1
fak n = n * fak (n-1)
calcCatalan n = fak (2*n) /  (fak (n+1) * fak n)
map' f [] = []
map' f (x:xs) = f x : map f xs
firstNCatalan n = map' calcCatalan [0..n]
main = print (firstNCatalan 12)
```