First class funktioner: Når et sprog behandler funktioner som first-class-citizens, dvs værdier. Funktionen kan gemmes i en variabel, kan være en property i et objekt, sendes/returneres som argument/returværdi fra en anden funktion.

En funktion som returnerer eller modtager en anden funktion, kaldes en ”higher-order-function”

Simpel forklaring: I stedet for at ændre variabelværdier inde i en funktion, henter man værdien ved at kalde en anden funktion (higher order: funkt der returnerer ell modtager en anden funkt).

Haskell eksempel:

Funktion der modtager funktion som args:

```
times4 :: Int -> Int
times4 x = x * 4

listTimes4 = map times4 [1,2,3,4,5]
```

Lambda:

```
addThree :: (Num a) => a -> a -> a -> a
addThree x y z = x + y + z
```

Er lig med:

```
addThree' :: (Num a) => a -> a -> a -> a
addThree' = \x -> \y -> \z -> x + y + z
```

Elm eksempler:

multThree : number -> number -> number -> number

multThree x y z = x * y *z

```
multTwoWithNine = multThree 9 <function> : number -> number -> number >
multTwoWithNine 2 3 54 : number
```

```
9 * ? * ? -> 9 * 2 * 3 = 54
```

Lambda:

```
\arguments -> returnedValue

> List.filter (\num -> num > 1) [1,2,3]
[2,3] : List number
```

JS eksempler:

1. Assign function to variable

```
Function square(x) {
        return x * x;
}

var f = square
```
console.log(f(5)) = 25

2. Function as arg to another function

```
function my_map(func, ar_list) {
        result = []
        for(var i = 0; i < arg_list.length; i++) {
            result.push(func(i))
        }
        return result;
}

var squares = my_map(square, [1,2,3,4,5])
```

console.log(squares) = [1,4,9,16,25]

3. Return function from function

```
function html_tag(tag){

        function wrap_text(msg){

           console.log('<' + tag + '>' + msg + '</' +tag + '>')

        }

return wrap_text

}

print_h1 = html_tag('h1')

print_h1('Headline!')
```

result: "<h1>Headline!</h1>