***Higher order functions***

- Er zijn twee manieren van software maken. Of je maakt het zo simpel dat er *duidelijk geen tekortkomingen* zijn, of je maakt het zo complex dat *er geen duidelijke tekortkomingen zijn.*

Groot programma 🡪 slecht begrijpbaar 🡪 veel menselijke fouten mogelijk 🡪 bugs (moeilijk verstopt).

**Abstractions** = verbergt detail en zorgt ervoor dat we op een hoger en abstracter level over code problemen kunnen praten. De juiste hoeveelheid hiervan is goed.

**Higher order functions** = functions die andere functions nodig hebben om te werken. Bijv.:

*greaterThan (n) => {*

*return m => m > n;*

*}*

*let greaterThan10 = greaterThan(10);*

*console.log(greaterThan10(11));*

*// → true*

- Ook kun je functies als parameter doorgeven. Zo kun je bijvoorbeeld een functie schrijven die verschillende functies kan toetsen. Deze functie is puur, omdat het een nieuwe array maakt met de items die de toets halen, in plaats van de orginele aanpassen:

*function filter(array, test) {*

*let passed = [];*

*for (let element of array) {*

*if (test(element)) {*

*passed.push(element);*

*}*

*}*

*return passed;*

*}*

*console.log(filter(SCRIPTS, script => script.living));*

*// → [{name: "Adlam", …}, …]*

- Zo kun je bijvoorbeeld twee nieuwe functies als parameters toevoegen:

*function map(array, transform) {*

*let mapped = [];*

*for (let element of array) {*

*mapped.push(transform(element));*

*}*

*return mapped;*

*}*

*let rtlScripts = SCRIPTS.filter(s => s.direction == "rtl");*

*console.log(map(rtlScripts, s => s.name));*

*// → ["Adlam", "Arabic", "Imperial Aramaic", …]*

- Eng ingewikkelde code: **reduce** om via een functie het totaal op te tellen:

*function reduce(array, combine, start) {*

*let current = start;*

*for (let element of array) {*

*current = combine(current, element);*

*}*

*return current;*

*}*

*console.log(reduce([1, 2, 3, 4], (a, b) => a + b, 0));*

*// → 10*