PS Funktionale Programmierung, SS 2024 – Blatt 3

1. a) Write a procedure *min-fx-gx* that takes two numerical procedures *f* and *g* and a number *x* as input, and that returns the minimum of applying *f* to *x* and *g* to *x*. Note: you may use *min*).

Example:
(min-fx-gx square cube -1) → -1 …because (`square -1)` is `1` and (`cube-1)` is `-1`
(min-fx-gx square cube 2) → 4

b) Then generalize these examples so that the procedure you apply to the results of *f* and *g* is a parameter too. The name of the procedure shall be *combine-fx-gx*.
Example:
(combine-fx-gx min square cube -1) → -1
(combine-fx-gx max square cube -1) → 1

2. Consider the following procedure:

```
(define (f g)
     (g 5))
```

First think, then try:

```
(f +)
(f square)
(f (lambda (x) (* x (+ x 2))))
(f f)
```

Explain the behavior (with comments)!

3. A function *f* is defined by the rule that $f(n) = n$ if $n<3$ and $f(n) = f(n - 1) + 2f(n - 2) + 3f(n - 3)$ if $n \geq 3$.

a) Write a procedure ***fr*** that computes *f* by means of a *recursive process*.
b) Write a procedure ***fi*** that computes *f* by means of an *iterative process*.

4. Hint: Make use of the ability to have functions as return values. Use lambda.

a) Define a procedure *twice* that takes a procedure of one argument and returns a procedure that applies the original procedure twice. E.g. *((twice square) 4)* → *256*.

b) Define a procedure *comp* that implements composition: The composition *f* after *g* is defined to be the function $x \mapsto f(g(x))$. The functions *f* and *g* shall be functions that have one argument only.
Examples:

```
((comp cube inc) 2)     ;27   (first increment, then cube: (2+1)³)

((comp inc cube) 2)     ;9    (first cube, then increment: 2³+1)
```

5. Be sure to understand the self-made *mycons*, *mycar*, *mycdr* functions. Define a data abstraction for representing *complex numbers* and implement a function *(add-complex c1 c2)* that returns the sum of two complex numbers. Do not use built-in pairs, but use the self-made mycons/mycar/mycdr from the lecture. Also write a *print-complex* function that outputs a nice representation of a complex number.