Aufgabe 1

```
def f(n):
    if n == 0:
        return 3
    else:
        return 2 + f(n-1)

print(f(4))
```

$$
\begin{aligned}
f(4) &= 2 + f(3) \\
&= 2 + (2 + f(2)) \\
&= 2 + (2 + (2 + f(1))) \\
&= 2 + (2 + (2 + (2 + f(0)))) \\
&\overset{*}{=} 2 + (2 + (2 + (2 + 3))) \\
&= 2 + (2 + (2 + 5)) \\
&= 2 + (2 + 7) \\
&= 2 + 9 \\
&= 11
\end{aligned}
$$

Aufgabe 2

```
def f(n):
    if n > 0:
        return n + f(n-2)
    else:
        return 7

print(f(5))
```

$$
\begin{aligned}
f(5) &= 5 + f(3) \\
&= 5 + (3 + f(1)) \\
&= 5 + (3 + (1 + f(-1))) \\
&\overset{*}{=} 5 + (3 + (1 + 7)) \\
&= 5 + (3 + 8)) \\
&= 5 + 11 \\
&= 16
\end{aligned}
$$

Aufgabe 3

```
def f(n):
    if n < 2:
        return n
    else:
        return f(n-1)+f(n-2)

print(f(5))
```

$$
\begin{aligned}
f(5) &= f(4) + f(3) \\
&= (f(3) + f(2)) + (f(2) + f(1)) \\
&= ((f(2) + f(1)) + (f(1) + f(0))) + ((f(1) + f(0)) + 1) \\
&= (((f(1) + f(0)) + 1) + (1 + 0)) + ((1 + 0) + 1) \\
&= (((1 + 0) + 1) + 1) + (1 + 1) \\
&= ((1 + 1) + 1) + 2 \\
&= (2 + 1) + 2 \\
&= 3 + 2 \\
&= 5
\end{aligned}
$$

Bemerkung: Die rekursiv definierte Funktion f definiert die *Fibonacci-Folge*

1, 1, 2, 3, 5, 8, 13, 21, ...

in der jede Zahl, abgesehen von der ersten, die Summe ihrer beiden unmittelbaren Vorgänger ist.

Die rekursive Definition ist äusserst ineffizient, da dieselben Rechnungen wiederholt ausgeführt werden.

Eine bessere Implementierung würde z. B. bereits berechnete Resultate in einer Tabelle zwischenspeichern.

Aufgabe 4

```
def f(n):
    if n > 4:
        return 3*f(n-2)
    elif n > 1:
        return 2*f(n-1)
    else:
        return 5

print(f(7))
```

$$
\begin{aligned}
f(7) &= 3 \cdot f(5) \\
&= 3 \cdot (3 \cdot f(3)) \\
&= 3 \cdot (3 \cdot (2 \cdot f(2))) \\
&= 3 \cdot (3 \cdot (2 \cdot (2 \cdot f(1)))) \\
&\overset{*}{=} 3 \cdot (3 \cdot (2 \cdot (2 \cdot 5))) \\
&= 3 \cdot (3 \cdot (2 \cdot 10)) \\
&= 3 \cdot (3 \cdot 20) = 3 \cdot 60 = 180
\end{aligned}
$$