Function / operator overloading

→ compiler muss wissen welche Funktion gemeint ist wenn er eine antrifft.

Funktions signatur wird eindeutig bestimmt durch
- ↳ Name der Funktion
- ↳ Anzahl der Argumente
- ↳ Datentypen der Argumente
- ↳ Reihenfolge der Argumente

NICHT durch
- ↳ Datentyp des return value
- ↳ Namen der Argumente

⇒ "overloading" ≙ mehrere Fkt mit gleichem Namen
d.h. unterscheidung durch Typ / Anzahl Argumente

Operator overloading

→ operatoren können als member und als non-member implementiert werden. Davon abhängig ist z.B. die Anzahl Argumente

→ Wie werden die op tatsächlich evaluiert?

(cppreference zu operator overloading)
https://en.cppreference.com/w/cpp/language/operators

Expression	As member function	As non-member function
@a	(a).operator@ ( )	operator@ (a)
a@b	(a).operator@ (b)	operator@ (a, b)

d.h. für @ ≙ + (≙ Addition)
haben wir a + b
- ↳ a.operator+(b) als member
- ↳ operator+(a, b) als non-member

→ In member functions haben wir implizit immer auch den this ptr als Argument dabei.

Hinweis zur Aufgabe Complex: (Bsp. zu Operator Overloading)

→ Ihr könnt Operatoren wie +, -, * etc immer innerhalb oder ausserhalb der Klasse / Struct deklarieren.

(Tendenziell bei Structs eher ausserhalb, bei classes eher innerhalb.)

→ Ausserhalb der Klasse definierte operatoren:

```
for struct Complex {
        double real;
        double imaginary;
    }                                  // per default sind alle member eines Structs public

wir müssen nicht mit :: sagen, in welcher Klasse wir sind
Complex operator+ ( const Complex& a, const Complex& b ){
    Complex result;
    result.real = ...
    result.imaginary = ...
    return result;
}
```

darum können wir hier auch von aussen darauf zugreifen

⇒ hier ist operator+ kein member d.h.
z1 + z2 wird evaluiert als
operator+ (z1, z2)

→ Innerhalb der Klasse

```
for class Complex {
    public:
        double real;
        double imaginary;
        // deklaration von operator+(const Complex & other)
};
```

// per default ist in einer Klasse alles private, habe es hier einfach der Einfachheit public gemacht.

```
Complex   Complex::operator+(const Complex& other){
            Complex result;
            result.real = real + other.real
            :
            :
}
```

(*) dieses real ist das real, welches zu der Instanz gehört, auf/mit welcher .operator+() aufgerufen wird. (Hier z1)

⇒ hier ist operator+ ein member (auch wenn wir die Fkt/ den Op ausserhalb definieren (out-of-class definition) so haben wir sie ja in der Class deklariert.)

z1 + z2 wird evaluiert als

z1.operator+(z2)
(*)