

## Wiederholung

---

```
for ( int x = 0; x < 100; x = x+1 ) {  
    // Anweisungen  
    // ...  
}
```

- Speicherplatz für einen veränderbaren Wert eines bestimmten Datentyps
  - Datentypen sind z. B. `int`, `double`, `boolean`
- Wichtige Begriffe
  - Deklaration
  - Initialisierung
  - Wertzuweisung
  - Gültigkeitsbereich

```
int x;  
double y = 1.234567;  
boolean zweiGroesserEins = true;  
y = y * (-0.75);
```

## Methoden Teil 2

---

## Aufgabe 1

- Schreibe eine Methode `berechneC()`, die als Parameter die Länge der beiden Katheten  $a$  und  $b$  eines rechtwinkligen Dreiecks erwartet und daraus mit dem Satz des Pythagoras die Länge der Hypotenuse  $c$  berechnet
  - Überlege Dir, welche Parameter erforderlich sind und welchen Datentyp diese haben sollten
  - Die Methode `Math.pow(double basis, double exponent)` erlaubt es, Potenzen zu berechnen
  - Die Methode `Math.sqrt(double radiant)` erlaubt es, die Quadratwurzel einer (nicht-negativen) Zahl zu berechnen
- Rufe `berechneC()` innerhalb von `setup()` auf.
  - Fällt Dir ein Problem auf?

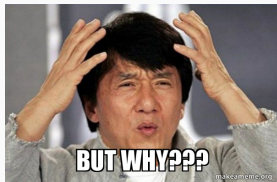
- Bisher: Zwei Arten von Methoden
  - Ohne Parameter
  - Mit Parameter(n)
- Problem dabei:
  - Wie kann der Aufrufer ein Ergebnis zurück bekommen?

## Vielleicht so?

```
void berechneC(double a, double b, double c) {  
    c = Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));  
}
```

```
void setup() {  
    double ergebnis = 0.0;  
    berechneC(3.5, 7.2, ergebnis);  
}
```

Geht so nicht!!!





- Parameter in Java werden mittels “call by value” an die Methode übergeben
  - Das bedeutet, eine Methode bekommt nur Kopien der übergebenen Werte
  - Veränderungen an den Parametern innerhalb der Methode haben keine Auswirkungen auf den Wert im Aufrufer
- Konkret:
  - Die Zuweisung an den Parameter `c` innerhalb von `berechneC( )` hat keine Auswirkungen auf die Variable `ergebnis`
  - Die Variable `ergebnis` aus dem letzten Beispiel hat immer den Wert `0.0`
- Lösung:
  - Methoden mit Rückgabewert

Definition (Beispiel):

```
int berechneWas() {  
    // Anweisungen  
    // ...  
    return ergebnis;  
}
```

Aufruf (z. B. in `setup()`):

```
int x = berechneWas();
```

**int statt void** Datentyp des zurückzugebenden Ergebnisses (könnte also auch ein anderer Datentyp als **int** sein)

**return ergebnis;** Rückgabe eines Wertes - in diesem Fall der Wert einer Variable mit dem Namen **ergebnis**. Mit dem Aufruf von **return** endet die Methode sofort!

```
int addiere(int a, int b) {  
    return a+b;  
}
```

```
double quadriere(double a) {  
    return a*a;  
}
```

```
int gibMirFuenf() {  
    return 5;  
}
```

```
boolean istNichtWahr() {  
    return !true;    // ! bedeutet Negation  
}
```

- Verändere die Methode `berechneC()` aus Aufgabe 1 so, dass sie das berechnete Ergebnis zurück gibt
- Rufe die Methode aus `setup()` heraus auf und speichere den zurückgegebenen Wert in einer geeigneten Variable
- Gib den Wert dieser Variable mit `System.out.println()` aus

## Schleifen Teil 2

---

- Bisher: for-Schleife für eine festgelegte Anzahl von Wiederholungen
- Oft ist im Vorfeld nicht bekannt, wie viele Wiederholungen es geben wird
  - Wiederhole, bis eine Taste gedrückt wird
  - Wiederhole, solange das Ergebnis einer Berechnung kleiner/größer ist als eine zuvor festgelegte Schwelle
  - Wiederhole, solange das Programm läuft
  - ...
- Problem:
  - Die Struktur der for-Schleife passt schlecht zu solchen Szenarien

- Bisher: for-Schleife für eine festgelegte Anzahl von Wiederholungen
- Oft ist im Vorfeld nicht bekannt, wie viele Wiederholungen es geben wird
  - Wiederhole, bis eine Taste gedrückt wird
  - Wiederhole, solange das Ergebnis einer Berechnung kleiner/größer ist als eine zuvor festgelegte Schwelle
  - Wiederhole, solange das Programm läuft
  - ...
- Problem:
  - Die Struktur der for-Schleife passt schlecht zu solchen Szenarien
- Lösung:
  - while-Schleife

# Die while-Schleife

Beispiel:

```
double d = 5.21;
while ( d > 0.7 ) {
    // Anweisungen
    // ...
}
```

**while** Schlüsselwort, das den Beginn einer while-Schleife anzeigt.

**{ bzw. }** Beginn und Ende des Schleifenrumpfs, also der Anweisungen, die wiederholt werden sollen.

**d > 0.7** Bedingung. Legt fest, dass der Schleifenrumpf so lange wiederholt werden soll, wie diese Bedingung gültig ist.



- Auch die Rückgabewerte von Methoden können innerhalb der Bedingung verwendet werden, z. B.:

```
while ( berechneC(x,y) < 5.0 ) {  
    // Anweisungen  
    // ...  
}
```

`< bzw. >` Echt kleiner bzw. Echt größer

`<= bzw. >=` Kleiner gleich bzw. Größer gleich

`==` Gleich (Vorsicht: **Nicht mit = verwechseln!**)

`!=` Ungleich

## Aufgabe 3

- Gegeben ist folgende for-Schleife:

```
for ( int x = 0; x < 20; x++ ) {  
    System.out.println(x);  
}
```

Gib eine zu dieser for-Schleife äquivalente while-Schleife an.

- Zusatzaufgabe: Gegeben ist folgende while-Schleife:

```
boolean running = true;  
while ( running == true ) {  
    // Anweisungen  
    // ...  
}
```

Gib eine zu dieser while-Schleife äquivalente for-Schleife an.

## Aufgabe 4

- Schreibe ein Programm, das die kleinste natürliche Zahl  $n$  ermittelt, deren Quadrat größer als 50000 ist. Verwende eine while-Schleife zur Lösung.
- Zusatzaufgabe: Lagere den Algorithmus in eine eigene Methode aus.
  - Die gesuchte Zahl soll als Ergebnis zurückgegeben werden.
  - Die Grenze soll nicht mehr fest auf 50000 festgelegt sein, sondern als Parameter angegeben werden können.
- Zusatzaufgabe: Löse die Aufgabe mit einer for-Schleife