

Dart

Programmiersprache für Smartphones



Agenda

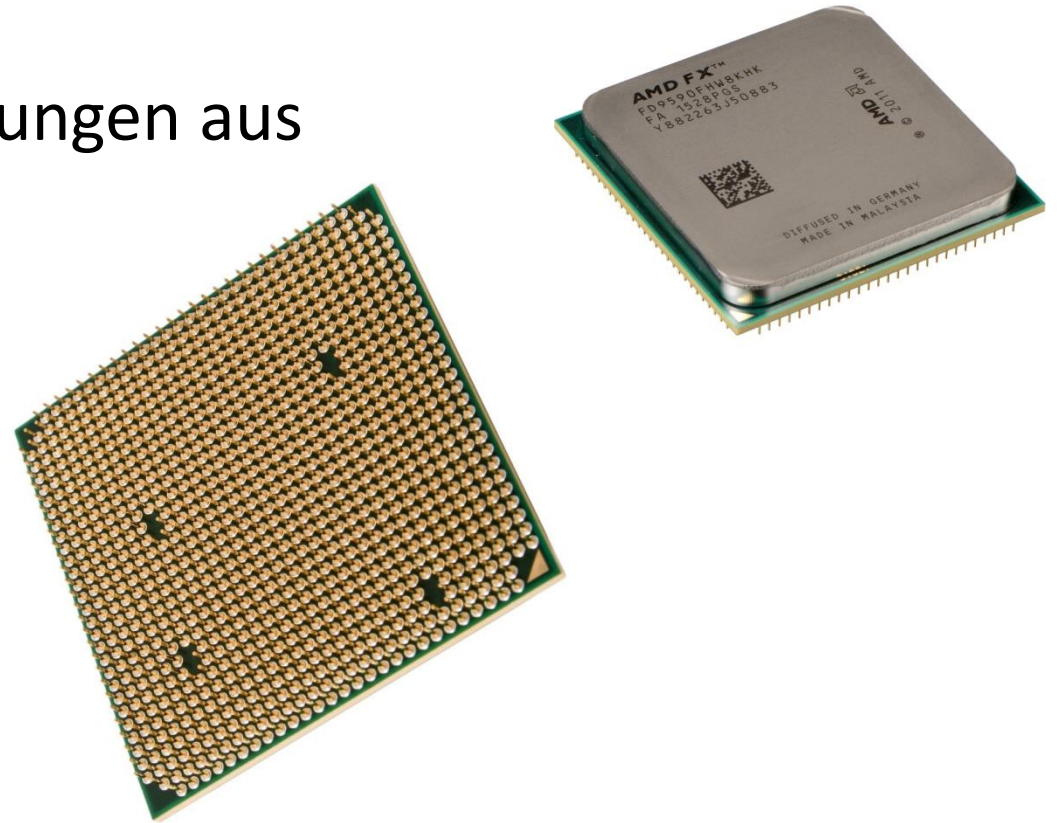


- Programmiersprachen
- Dart
 - Textausgabe
 - Kommentare
 - Rechnen
 - Bibliotheken
 - Texte (Strings)
 - Wiederholungen
 - Wahrheitswerte
 - Verzweigungen
 - Listen
- Map
- Methoden
- Funktionen
- Android Studio
 - Rechtschreibprüfung
 - Code Formatierung
 - Live Templates

Programmiersprachen - Maschinensprache

- Prozessor versteht Maschinensprache
- Prozessoren führen Befehle / Berechnungen aus
- Maschinensprache sind Zahlen
 - Meist Hexadezimal

Maschinencode (hexadezimal)
55
48 89 E5
C7 45 FC 02
C7 45 F8 03



Programmiersprachen - Assembler

- Hardwarenah
- Für tief eingestiegene Programmierer lesbar
- Befehle (**blau**) und Daten (**braun**)
 - mov: Daten verschieben
 - r: Register
 - DWORD: Anzahl der Bits

Maschinencode (hexadezimal)	zugehöriger Assemblercode
55	<code>push rbp</code>
48 89 E5	<code>mov rbp, rsp</code>
C7 45 FC 02	<code>mov DWORD PTR [rbp-4], 2</code>
C7 45 F8 03	<code>mov DWORD PTR [rbp-8], 3</code>

Programmiersprachen - C

- Programmiersprache von 1972
- Gliederung in Funktionen, Variablen, ...

Maschinencode (hexadezimal)	zugehöriger Assemblercode	zugehöriger C-Code
55 48 89 E5	<code>push rbp</code> <code>mov rbp, rsp</code>	<code>int main() {</code>
C7 45 FC 02	<code>mov DWORD PTR [rbp-4], 2</code>	<code>int a = 2;</code>
C7 45 F8 03	<code>mov DWORD PTR [rbp-8], 3</code>	<code>int b = 3;</code>

Programmiersprachen - Dart

- Hochsprache
 - leicht(er) verständlich
 - nicht so nah am Prozessor orientiert
- Kostenlos verfügbar
- Universalsprache („general purpose programming language“)
 - kann viele verschiedene Probleme lösen

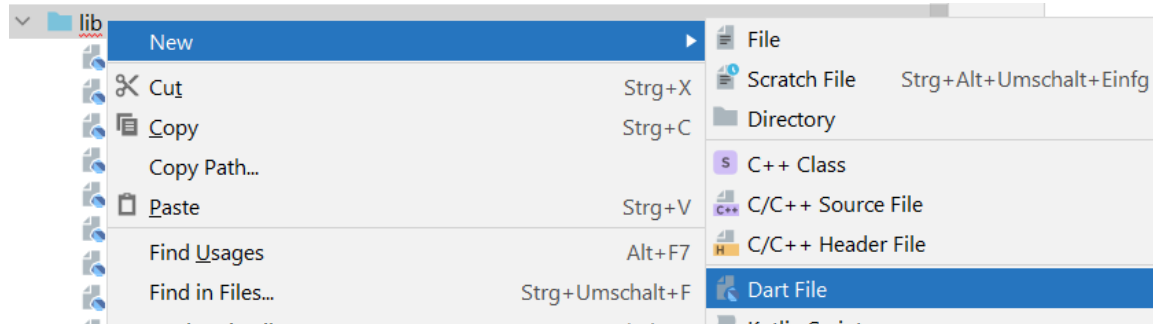


Dart - Dateiformat

- Programmiert wird im Textformat
 - Textdatei, UTF-8 Encoding
 - d.h. Sonderzeichen wie Smileys werden unterstützt
- Anweisungen werden mit **;** getrennt
 - bitte trotzdem nur eine Anweisung pro Zeile
- Einrückung ist empfohlen
 - der Lesbarkeit halber
 - syntaktisch jedoch nicht erforderlich
- Einstiegspunkt **void main() { }**

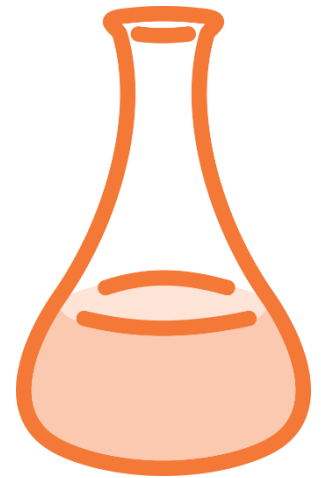
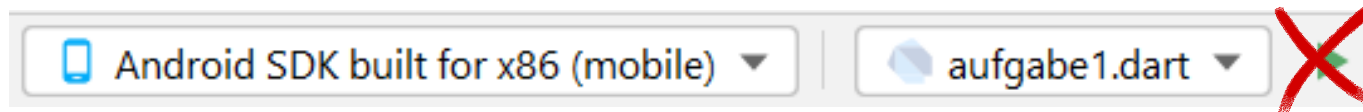
Dart - Ausgabe auf dem Bildschirm

- Lege eine neue Datei an: Aufgabe1.dart



- Textausgabe: **print("...");** oder **print('...');**
- Programmiere:

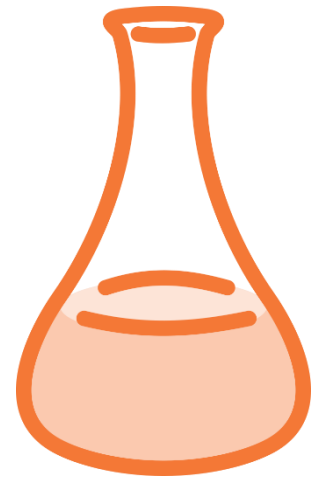
```
1 >> void main() {  
2     print("Hello Dart 😊");  
3 }
```



Dart - Kommentare

- Kommentare mit `//`
- Mehrzeilige Kommentare mit `/* ... */`

```
1  /*  
2   Dies ist unsere erste Aufgabe. Wir geben Text aus.  
3   Und das hier ist ein Kommentar, der nichts tut, außer die Aufgabe zu beschreiben.  
4  */  
5  ▶▶ void main() {  
6      // Die folgende Zeile gibt Text aus  
7      print("Hallo Dart 🍷👉🤖👉🤖");  
8  }
```



Android Studio: Rechtschreibprüfung

```
1  /*
2   Dies ist unsere erste Aufgabe. Wir geben Text aus.
3   Und das hier ist ein Kommentar, der nichts tut, außer die Aufgabe zu beschreiben.
4  */
5  void main() {
6   // Die folgende Zeile gibt Text aus
7   print("Hallo Dart 🍌🍌🍌");
8 }
```

Quelltext mit deutschen Kommentaren kommt in vielen Foren nicht gut an.

The screenshot shows the 'Ask a question' page on Stack Overflow. The 'Title' field contains 'I have a question for my Dart program'. The 'Body' field contains the same Dart code as shown in the first block, with the German comments highlighted in a red box. The right sidebar shows the 'Step 1: Draft your question' section with instructions and a list of steps: '1. Summarize the problem', '2. Describe what you've tried', and '3. Show some code'.

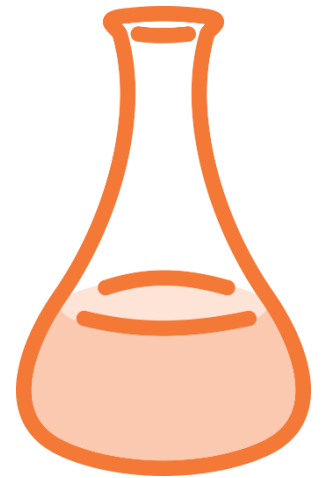
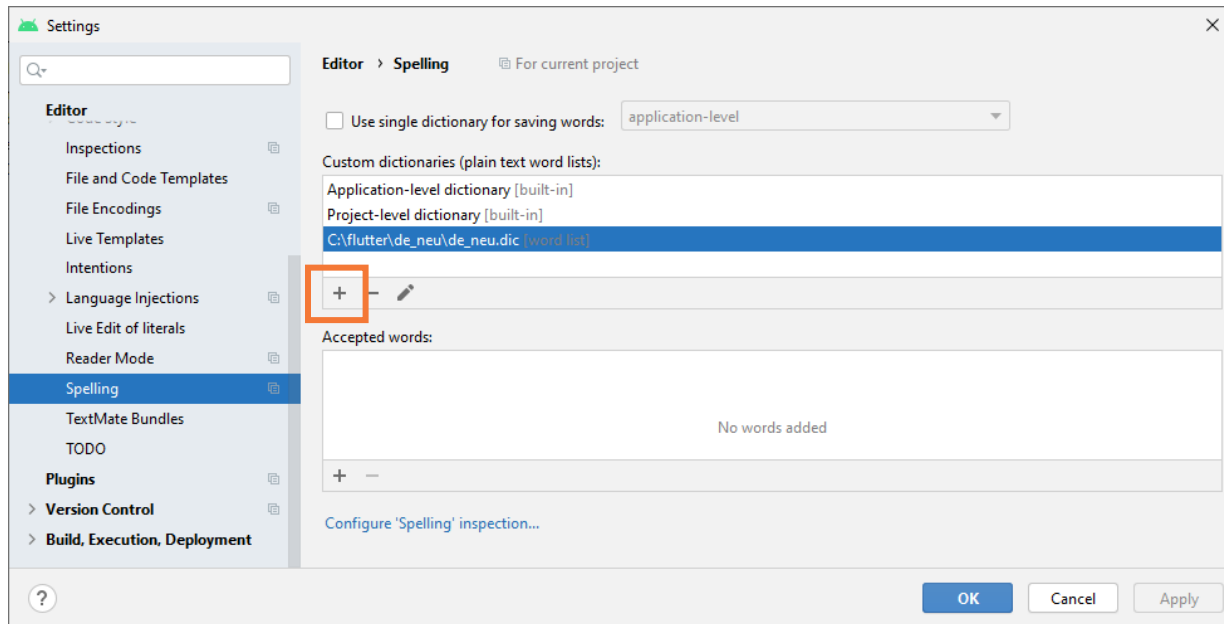
Android Studio: Rechtschreibprüfung

- Wörterbücher
- <http://www.winedt.org/dictASCII.html>

German (new sp)	Info	Dictionary	2039 KB	2005-01-06
German (old+new sp)	Info	Dictionary	1135 KB	1999-05-10
German (CH, old+new sp)	Info	Dictionary	1127 KB	2000-07-21

Android Studio: Rechtschreibprüfung

- File / Settings
- Editor/Spelling
- de_neu.dic hinzufügen



Android Studio: Rechtschreibprüfung

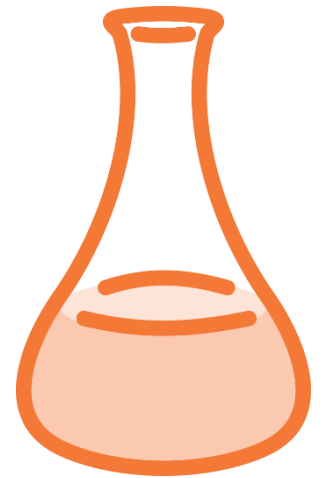
- Schon besser

```
1  /*
2   Dies ist unsere erste Aufgabe. Wir geben Text aus.
3   Und das hier ist ein Kommentar, der nichts tut, außer die Aufgabe zu beschreiben.
4  */
5  void main() {
6      // Die folgende Zeile gibt Text aus
7      print("Hallo Dart 🍷🍷🍷🍷");
8  }
```

Dart - Rechnen

- Variable: **var** *name* = *wert*;
- Mathematische Operatoren: **+**, **-**, *****, **/**
- Punkt vor Strich
- Automatische Erkennung des Zahlentyps bei **var**

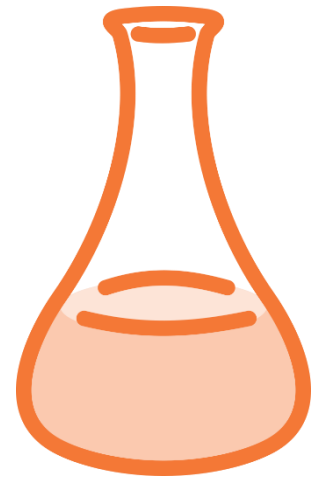
```
1 ► void main() {  
2     var x = 5;  
3     var ergebnis = x+x/5;  
4     print(ergebnis);  
5 }
```



Dart - Rechnen

- Ganzzahl Division: `~/`
- Rest (Modulo): `%`
- Klammersetzung: `(...)`

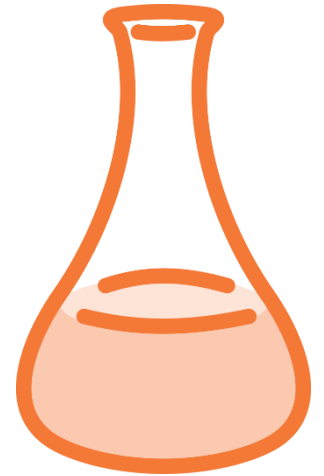
```
1 ▶ void main() {  
2     print(7/3);  
3     print(7~/3);  
4     print(7%3);  
5     print(7/(2+1));  
6 }
```



Dart - Rechnen

- Abkürzungen
 - **i++** bedeutet **i=i+1**
 - **i+=2** bedeutet **i=i+2**
 - **i*=2** bedeutet **i=i*2**
 - Dito: **--**, **-=**
 - Bitte nicht: **/=**, **~/=**, **%=**

```
1  >> void main() {  
2      var i=0;  
3      i++;  
4      i+=2;  
5      i*=2;  
6      print(i);  
7  }
```



Dart - Bibliotheken

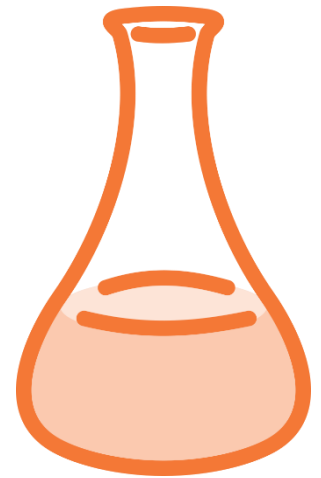
$$\frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\frac{x^2}{2}} dx \quad ?$$

- Hilfe!
- Bibliothek = Sammlung fertiger Funktionen
- Bibliothek einbinden: **import "bibliothek";**
- Immer dabei: **"dart:core"**
- Mathe-Bibliothek: **"dart:math"**
- Umwandlung von Daten: **"dart:convert"**
- ...

Dart - Bibliotheken

- Funktionen direkt importieren
- Vorteil: weniger zu tippen
- Nachteil: manche Namen sind dann schon vergeben
- Nachteil: man muss die Funktionen kennen

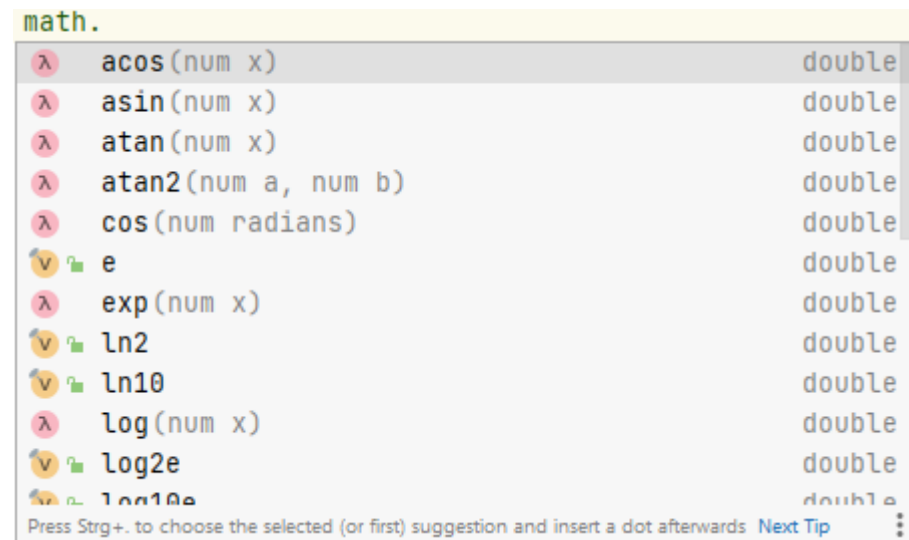
```
1 import "dart:math";  
2  
3 ► void main() {  
4     print(pow(3, 4));  
5     print(sqrt(2));  
6     print(e);  
7     print(pi);  
8 }
```



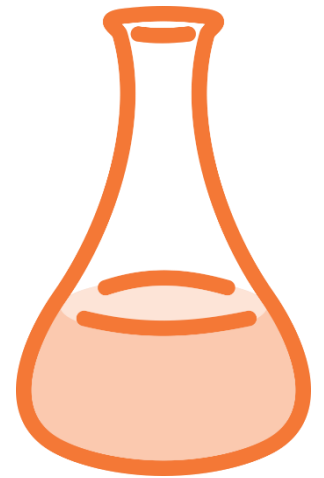
Dart - Bibliotheken

- Funktionen mit Alias importieren: **import "bibliothek" as name;**
- Nachteil: etwas mehr zu tippen
- Vorteil: IntelliSense

```
1 import "dart:math" as math;
2
3 void main() {
4     print(math.pow(3, 4));
5     print(math.sqrt(2));
6     print(math.e);
7     print(math.pi);
8 }
```

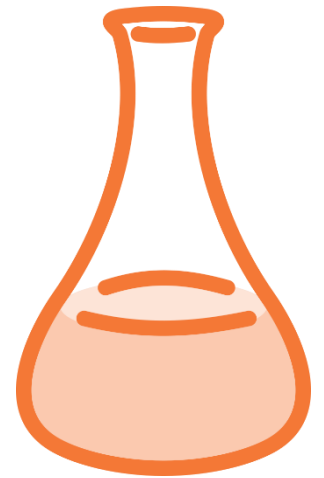


IntelliSense: Vorschläge beim Tippen



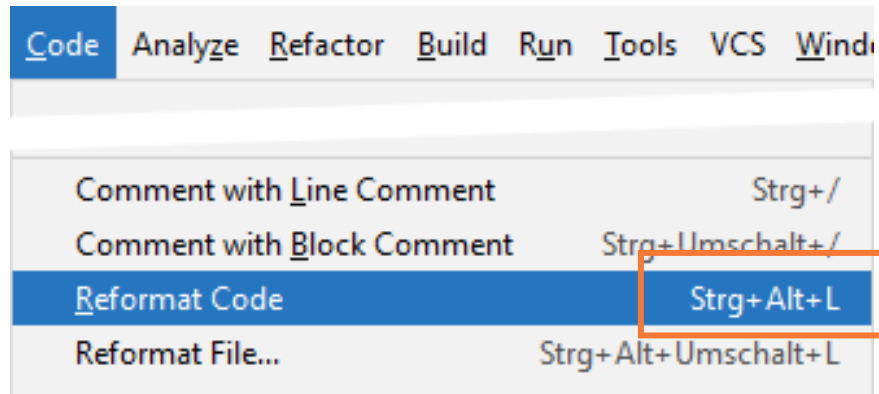
Dart - Aufgabe

Lasse $356 \cdot 4^3$ berechnen



Android Studio – Code Formatierung

- Code soll immer ähnlich aussehen
- Erhöht die Lesbarkeit
- Verringert Fehler
- Erleichtert Vergleichbarkeit



```
* Unbenannt links  
print (4 * 5 + 2 * 9 ~/ 6) ;  
  
* Unbenannt rechts  
print (4 * 6 + 2 * 8 ~/ 6) ;
```

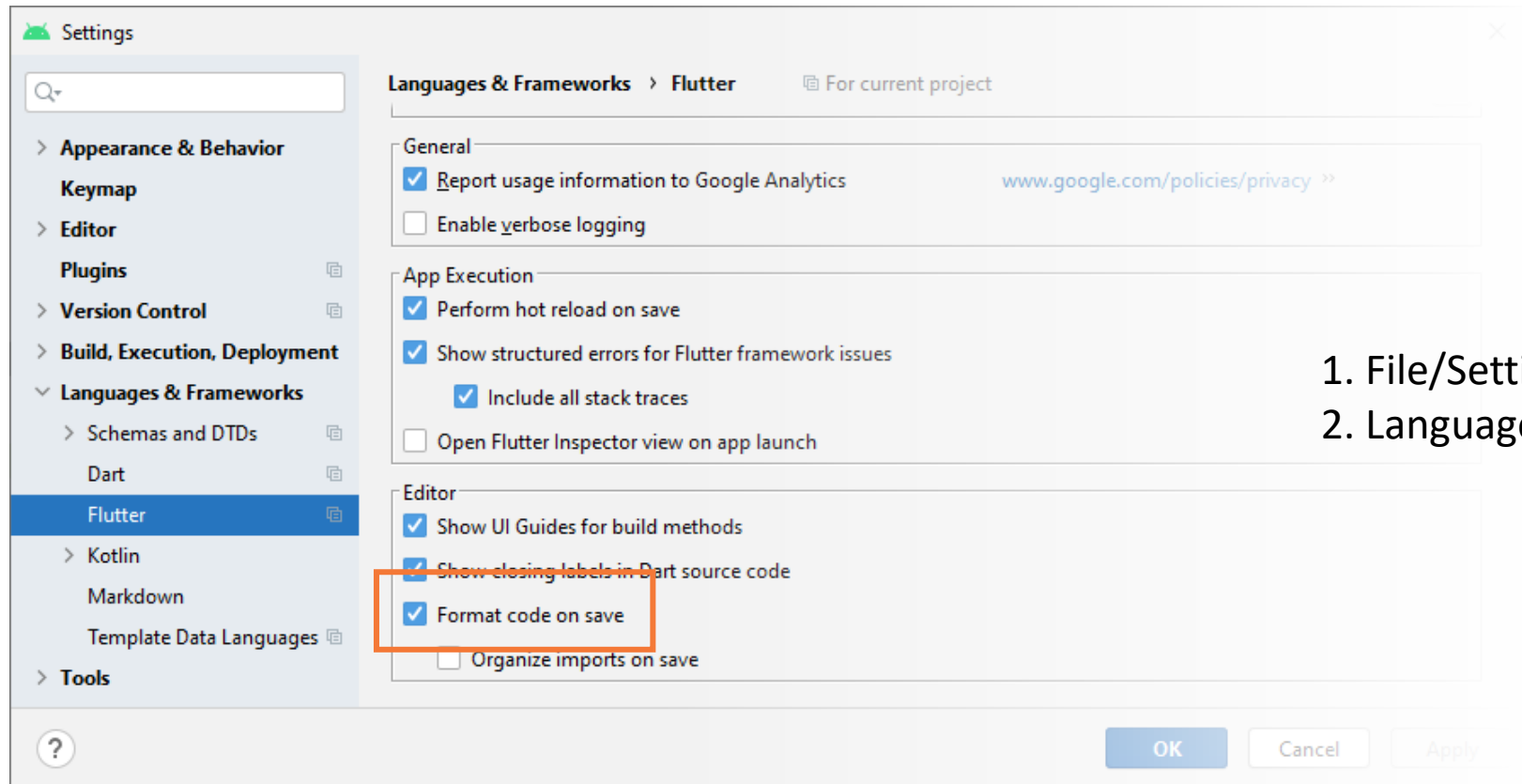
Unterschiede bei anderer Formatierung

```
* Unbenannt links  
print (4 * 5 + 2 * 9 ~/ 6) ;  
  
* Unbenannt rechts  
print (4 * 6 + 2 * 8 ~/ 6) ;
```

Unterschiede bei gleicher Formatierung

Android Studio – Code Formatierung

- Noch einfacher: beim Speichern formatieren



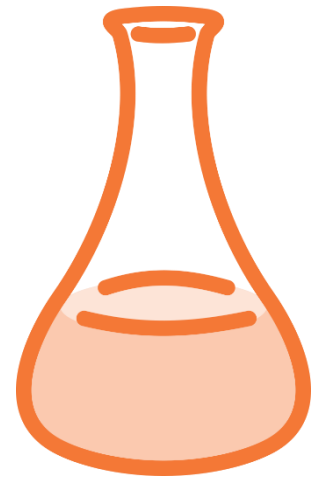
1. File/Settings

2. Languages and Frameworks / Flutter

Dart - Strings

- Variablen können nicht nur Zahlen sein sondern auch Text
- In Anführungszeichen ("...") („quotation marks“, “double quotes“)
- In Hochkomma ('...') („apostrophe“)

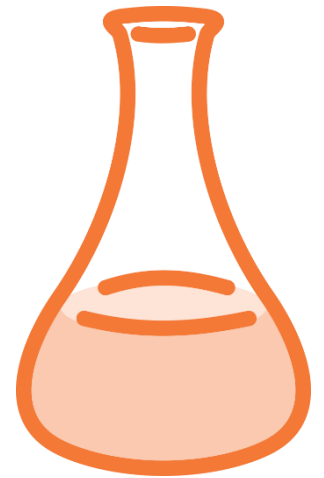
```
1 ▶ void main() {  
2     var text = "Hello Dart '😊'";  
3     print(text);  
4     var hello = 'Hello Dart "😊"';  
5     print(hello);  
6 }
```



Dart - Strings

- Sonderzeichen mit sog. „Escaping“
- Neue Zeile: `\n`
- Backslash: `\\`
- Anführungszeichen, Apostroph: `\"` , `\'`
- Sonderzeichen aus Zeichentabelle (Unicode-Tabelle): `\uhhhh`

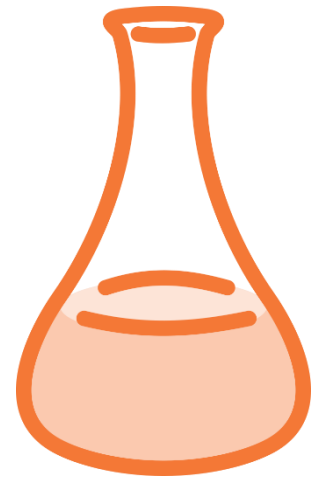
```
1 >> void main() {  
2     var quotes = "Sonderzeichen: \' \" \\ \";  
3     var linebreak = "Line\nBreak";  
4     var mitutoyo = "\u30DF\u30C4\u30C8\u30E8";  
5     print(quotes);  
6     print(linebreak);  
7     print(mitutoyo);  
8 }
```



Dart - Strings

- „Rechnen“ mit Text
- Zerteilen mit **.substring(*anfang*, *ende*)**
- Großbuchstaben mit **.toUpperCase()**
- ...

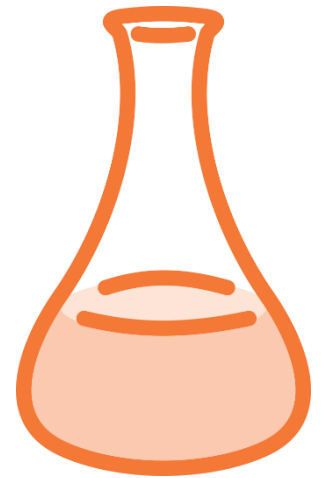
```
1  >> void main() {  
2      var Ha = "Ha";  
3      var ll = "l"*2;  
4      var ooo = 'oh'*10;  
5      print(Ha+ll+ooo);  
6  
7      var all = (Ha+ll+ooo).substring(1,4);  
8      print(all.toUpperCase());  
9      var raten = Ha+ll+ooo.substring(1,4);  
10     print(raten);  
11 }
```



Dart - Strings

- Umwandlungen

```
1  >> void main() {  
2      var text = "1";  
3      print(text*10);           // Diese 1 verhält sich wie ein Text  
4      var zahl = int.parse(text);  
5      print(zahl*10);          // Diese 1 verhält sich wie eine Zahl  
6      var wiederText = zahl.toString();  
7      print(wiederText*10);     // Diese 1 verhält sich wieder wie ein Text  
8  }
```

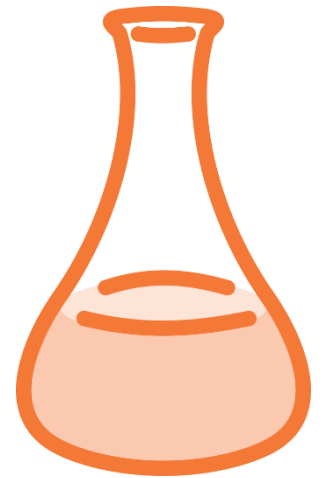


Dart - Strings

- Variablen in Strings einfügen

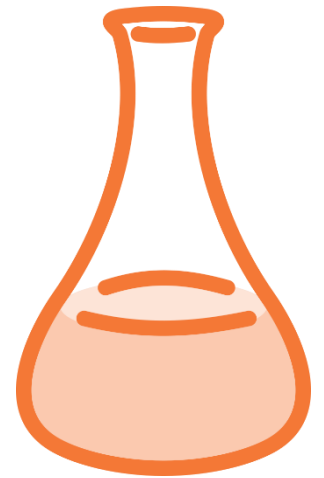
var name = "Text **\${variable}** Text";

```
1 ▶ void main() {  
2     var name = "John";  
3     print("Hallo ${name}");  
4 }
```



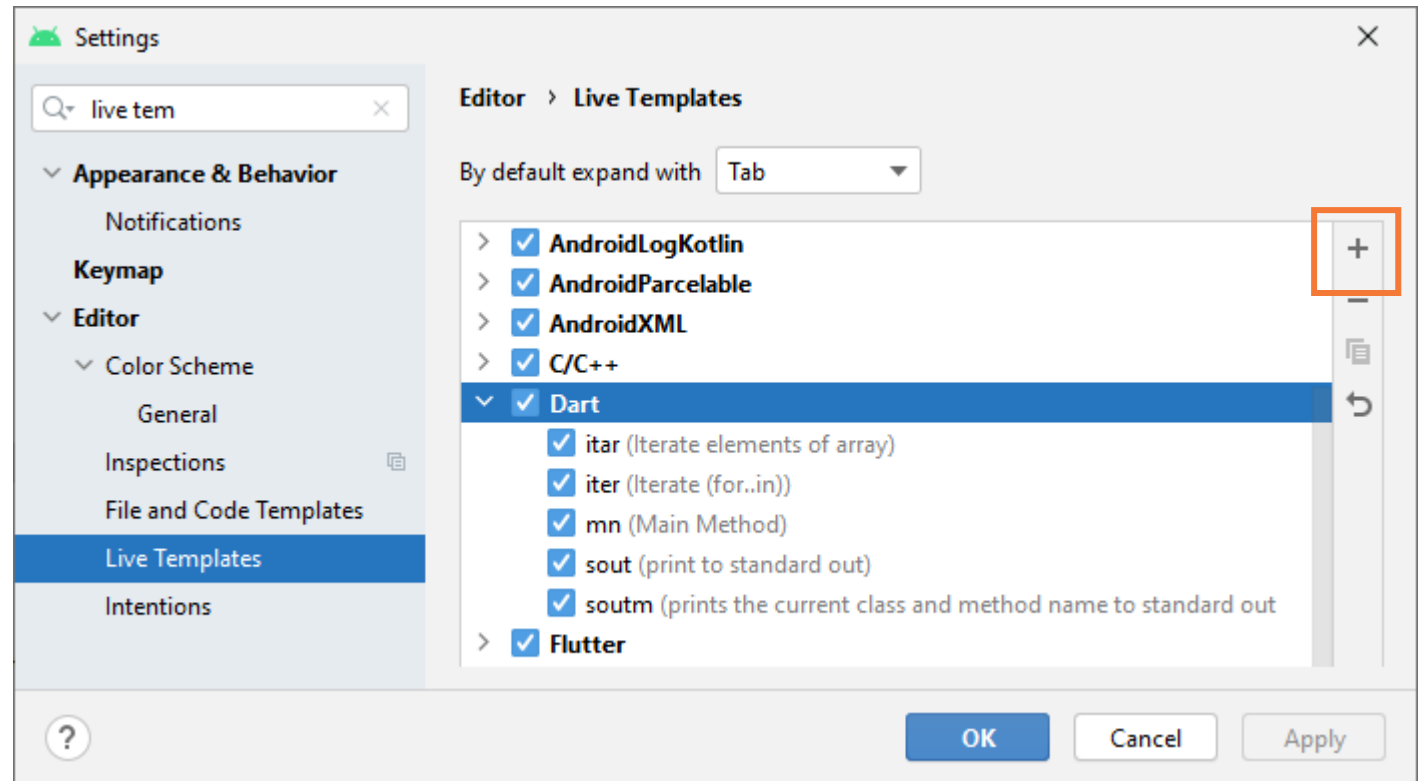
Dart – Strings - Aufgabe

- Was ist das Quadrat der mittleren beiden Ziffern von 36^2 ?
- $36^2 \rightarrow XYYX \rightarrow YY \rightarrow YY^2$
- Löse die Aufgabe so, dass sie möglichst einfach für beliebige andere Zahlen angepasst werden kann
- Zulässige Annahme
 - Die Ausgangszahl liegt zwischen 32 und 99
 - d.h. es ergibt sich immer eine vierstellige Zahl



Android Studio - Live Templates

- Immer wieder das gleiche tippen?
- Lösung: Live Templates
- File/Settings
- Editor/Live Templates
- Dart



Android Studio - Live Templates

Abbreviation: Description:

Template text:

```
void main() {  
$END$  
}
```

Options

Expand with

- ☒ Reformat according to style
- ☒ Shorten FQ names

Applicable in [Change](#) ▾

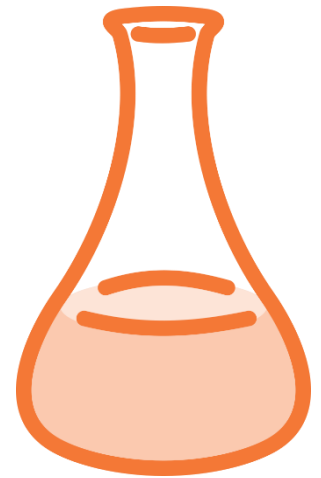
Dart -Wiederholungen

- Für eine bekannte Anzahl Durchläufe:

for (var *zähler* = *anfang*; *zähler* < *ende*; *zähler*++) { ... }

- Zähler: oft i, j, k

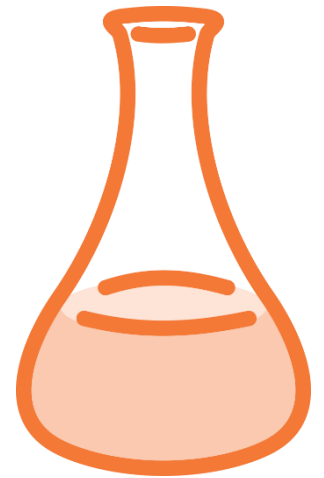
```
1  >> void main() {  
2      for (var i = 0; i < 100; i++) {  
3          print(i);  
4      }  
5  }
```



Dart -Wiederholungen

- Für eine unbekannte Anzahl Durchläufe:
while (*bedingung*) { ... }

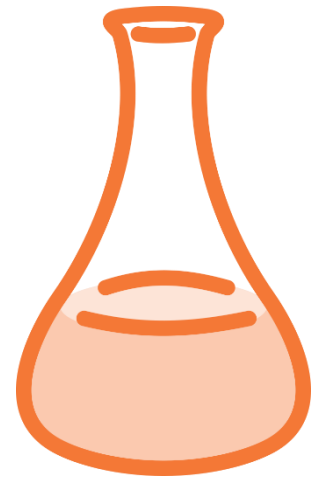
```
1  ▶ void main() {  
2      var expo = 1.0007;  
3      while (expo < 1000) {  
4          expo = expo * expo;  
5          print(expo);  
6      }  
7  }
```



Dart - Wahrheitswerte

- Aussagen können wahr (**true**) oder falsch (**false**) sein
- Operatoren
 - kleiner: **<**
 - kleiner oder gleich: **<=**
 - größer: **>**
 - größer oder gleich: **>=**
 - gleich: **==**
 - ungleich: **!=**

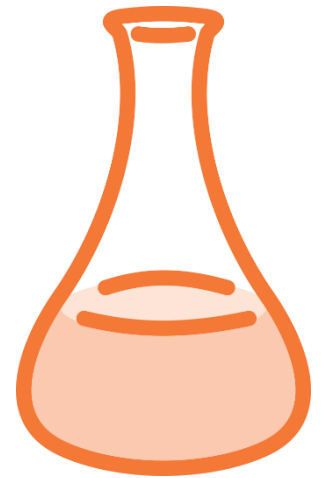
```
1  >> void main() {  
2      print(3 < 5); // true  
3      print(7 <= 7); // true  
4      print(9 > 11); // false  
5      print(13 == 15); // false  
6      print(17 != 19); // true  
7  }
```



Dart - Wahrheitswerte

- Aussagen können verknüpft werden
- Operatoren
 - und: **&&** (beide müssen wahr sein)
 - oder: **||** (mindestens eins muss wahr sein)
 - **&&** hat Vorrang vor **||**
 - Klammern möglich

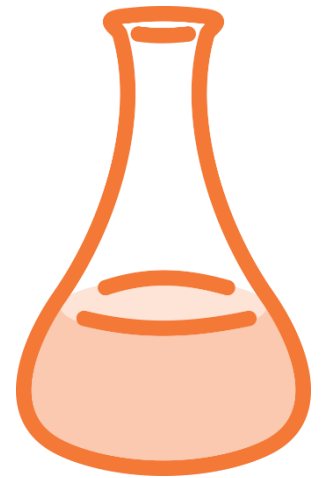
```
1  ▶ void main() {  
2      print(3 > 5 && 7 < 9);  
3      print(true && true);  
4      print(false || 11 != 13);  
5      print(true && true || false);  
6  }
```



Dart - Wahrheitswerte

Finde heraus, ob die Aussage $a \leq b \geq c$ wahr oder falsch ist für

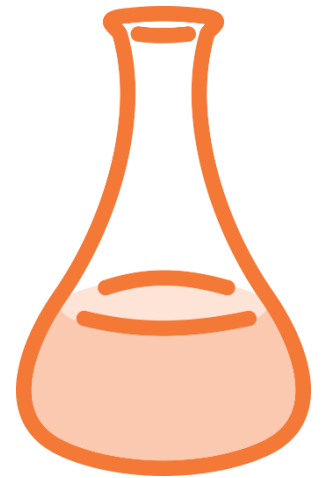
- $a=3, b=9, c=17$
- $a=1, b=2, c=2$



Dart - Verzweigungen

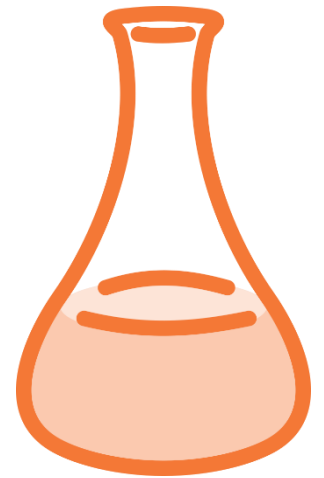
- if-Befehl
 - **if** (*Bedingung*) {
 // wenn *Bedingung* wahr ist
} **else if** (*Bedingung2*) {
 // wenn *Bedingung* nicht wahr aber *Bedingung2* wahr ist
} **else** {
 // wenn weder *Bedingung* noch *Bedingung2* wahr sind
}

```
1  >> void main() {  
2      if (3 < 7) {  
3          print("3 ist kleiner als 7.");  
4      } else {  
5          print("3 ist nicht kleiner als 7.");  
6      }  
7  }
```



Dart - Verzweigungen - Aufgabe

Wie viele Zahlen von 100 bis 999 enthalten die Ziffer 3?



Dart - Verzweigungen - Aufgabe

```
1  >> void main() {  
2      var x = 100;  
3      var y = 1000;  
4      var z = 0;  
5      while (x < y) {  
6          var xs = x.toString();  
7          if (xs.contains("3")) {  
8              z += 1;  
9          }  
10         x += 1;  
11     }  
12     print(z);  
13 }
```

```
1  >> void main() {  
2      var anzahl = 0;  
3      for (var zahl = 100; zahl <= 999; zahl++) {  
4          var text = zahl.toString();  
5          var ziffer1 = text.substring(0, 1);  
6          var ziffer2 = text.substring(1, 2);  
7          var ziffer3 = text.substring(2, 3);  
8          if (ziffer1 == "3" || ziffer2 == "3" || ziffer3 == "3") {  
9              anzahl++;  
10         }  
11     }  
12     print(anzahl);  
13 }
```

Dart - Listen

- Listen beinhalten viele Daten ohne dass jedes Datum einen eigenen Namen bekommen muss.

```
1  >> void main() {  
2      var zahl1 = 2;  
3      var zahl2 = 3;  
4      var zahl3 = 5;  
5      var zahl4 = 7;  
6      var zahl5 = 11;  
7      var zahl6 = 13;  
8      var zahl7 = 17;  
9      print(zahl1);  
10     print(zahl2);  
11     print(zahl3);  
12     print(zahl4);  
13     print(zahl5);  
14     print(zahl6);  
15     print(zahl7);  
16 }
```

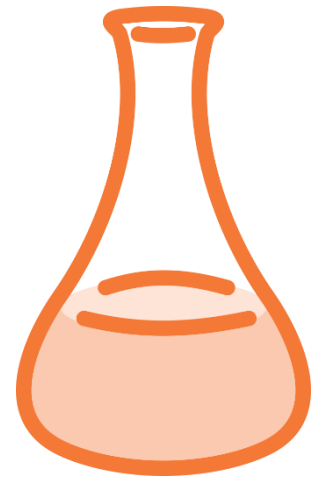
Wird irgendwann langweilig zum tippen

Dart - Listen

- Liste: **var *liste* = [*wert*, *wert*, ...];**
- Zur Liste gibt es eine Variante der For-Schleife:
for (var *name* in *liste*) { ... }

```
1  ▶ void main() {  
2      var list = [2, 3, 5, 7, 11, 13, 17];  
3      for (var zahl in list) {  
4          print(zahl);  
5      }  
6  }
```

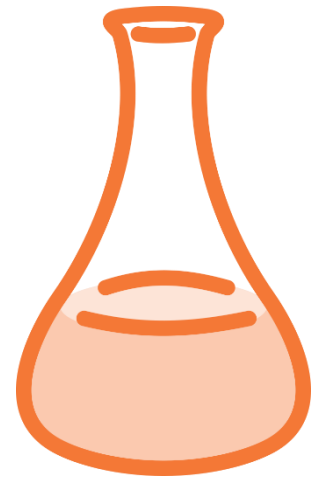
Irgendwie besser



Dart - Listen

- "Rechnen" mit Listen
- Aneinanderhängen: +

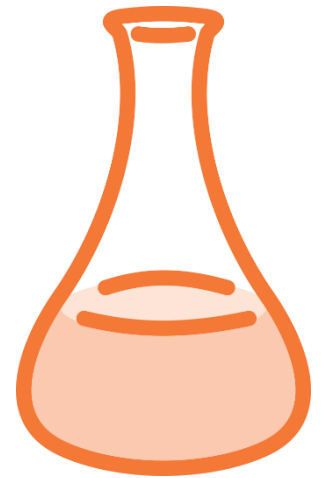
```
1  ▶▶ void main() {  
2      var list1 = [1, 2, 3];  
3      var list2 = [10, 20, 30];  
4      print(list1 + list2);  
5  }
```



Dart - Listen

- Element aus der Liste holen: `var name = liste[index];`
- Element in der Liste austauschen: `liste[index] = wert;`
- Die Zählung beginnt bei 0

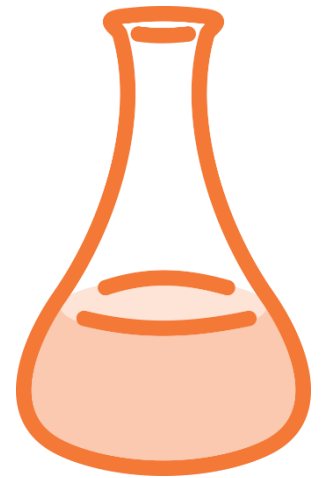
```
1  ▶ void main() {  
2      var list = [1, 2, 3, 10, 20, 30];  
3      print(list[3]);  
4      list[2] = 5;  
5      print(list);  
6  }
```



Dart - Listen

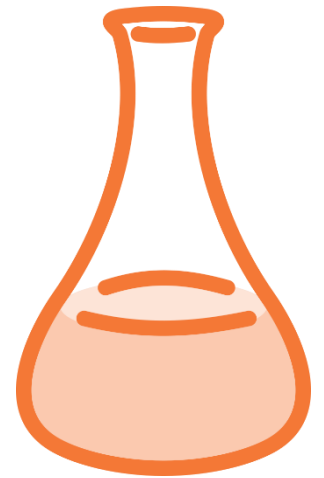
- Listen zerteilen: **var teil = liste.sublist(start, ende);**

```
1  >> void main() {  
2      var liste = [2, 4, 8, 16, 32, 64];  
3      var zwei = liste.sublist(0, 2);  
4      var vier = liste.sublist(2);  
5      print(zwei);  
6      print(vier);  
7  }
```



Dart - Listen Aufgabe

Gib die letzten drei Elemente der Liste [0,1,1,2,3,5,8,13] aus.



Dart - Map

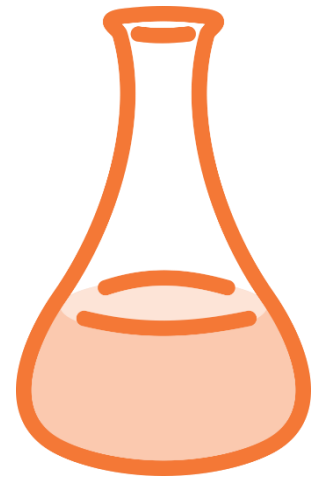
- Map: Abbildung
- Abbildung von einem Wert (Key) auf einen anderen (Value)

`var name = { key:value, ... };`

- Zugriff über den Key in Klammern:

`var value = name[key];`

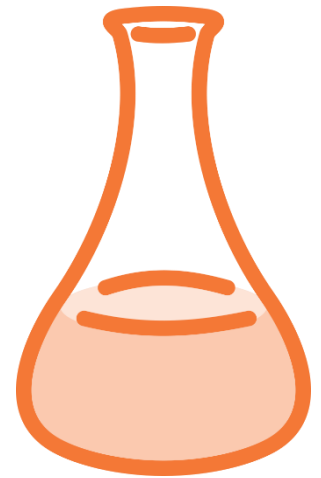
```
1  ▶ void main() {  
2      var dictionary = {"apple": "Apfel", "banana": "Banane"};  
3      var prices = {"apple": 3.49, "banana": 2.99};  
4      var item = "apple";  
5      print(dictionary[item]);  
6      print(prices[item]);  
7  }
```



Dart - Map

- Schleife für eine Map:
for (var pair in dictionary.entries) {
...
}

```
1  ▶▶ void main() {  
2      var dictionary = {"apple": "Apfel", "banana": "Banane"};  
3      for (var pair in dictionary.entries) {  
4          print("${pair.key}\t${pair.value}");  
5      }  
6  }
```

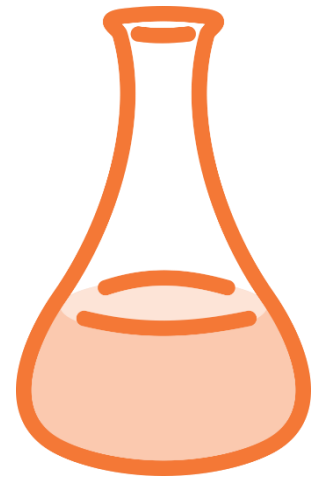


Dart - Methoden

- Methoden dienen der Wiederverwendung von Code

void *name*(**var** *argument*, **var** *argument2*, ...) { ... }

```
1  >> void main() {  
2      irre(1);  
3      irre(2);  
4      irre(3);  
5  }  
6  
7  void irre(var argument) {  
8      for (var i = 0; i < argument; i++) {  
9          print("Das ist irre");  
10         print("Stell Dir vor ich müsste das ständig kopieren");  
11     }  
12     print("-" * 10);  
13 }
```

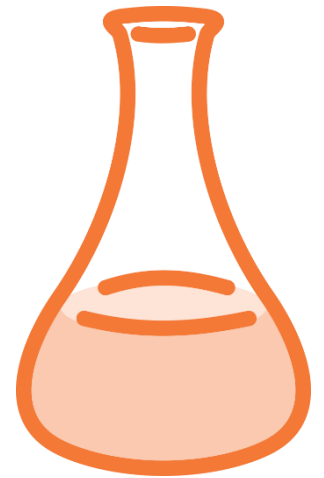


Dart - Methoden

- Bei Methoden ist es sinnvoll den Typ der Argumente genauer anzugeben als nur **var**

```
1  >> void main() {  
2      irre(1);  
3      irre(2);  
4      irre("5");  
5  }  
6  
7  void irre(var argument) {  
8      for (var i = 0; i < argument; i++) {  
9          print("Das ist irre");  
10         print("Stell Dir vor ich müsste das ständig kopieren");  
11     }  
12     print("-" * 10);  
13 }
```

Kein Fehler erkannt

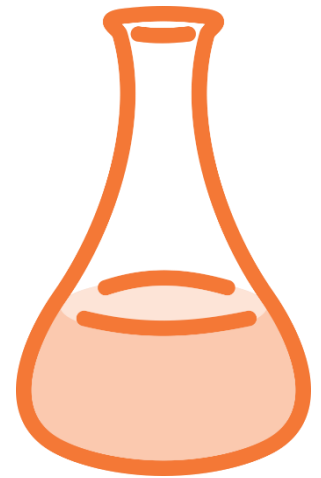


Dart - Methoden

- Mit **int** erkennt die Programmierung den Fehler schon vorher

```
1  >> void main() {  
2      irre(1);  
3      irre(2);  
4      irre("5");  
5  }  
6  
7  void irre(int argument) {  
8      for (var i = 0; i < argument; i++) {  
9          print("Das ist irre");  
10         print("Stell Dir vor ich müsste das ständig kopieren");  
11     }  
12     print("-" * 10);  
13 }
```

Fehler erkannt



Dart - Funktionen

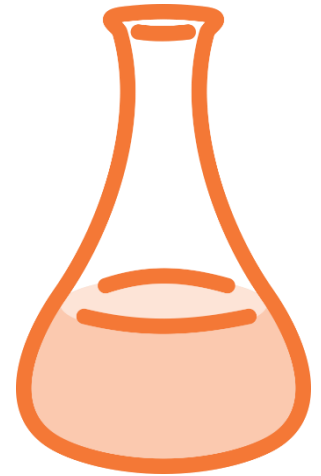
- Funktionen sind ähnlich wie Methoden, liefern aber ein Ergebnis
ergebnistyp *name*(*var argument*, *var argument2*, ...) {

...

return ergebnis;

}

```
1  >> void main() {  
2      var a = rechne(5, 4);  
3      print(a);  
4      var b = rechne(7, 3);  
5      print(b);  
6  }  
7  
8  int rechne(int i, int j) {  
9      var irgendwas = i * i + i * (i + j) + i * j + j * j;  
10     return irgendwas;  
11 }
```

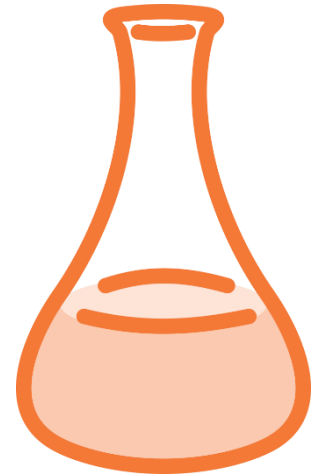


Dart - Funktionen

Programmiere eine Funktion,
die das gleiche Ergebnis liefert wie `math.pow()`

Zulässige Annahme:

- Nur natürliche Zahlen (\mathbb{N}_0)

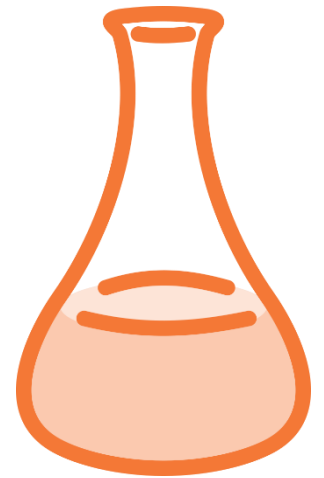


Dart - Funktionen

Schreibe eine Funktion,
die aus einer Liste mit Zahlen die kleinste heraussucht.

Zulässige Annahme:

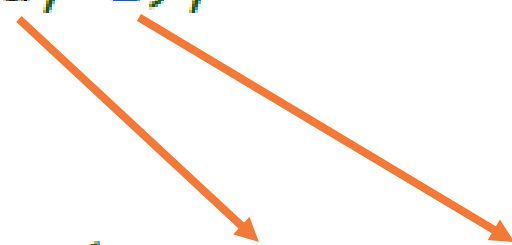
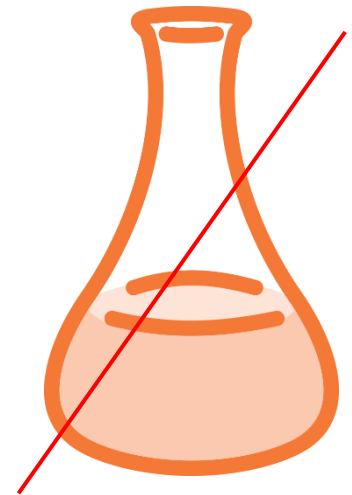
- Nur ganze Zahlen (\mathbb{Z})
- Mindestens eine Zahl in der Liste



Dart - Named Arguments

- Bisher: die Reihenfolge spielte eine Rolle.

```
1  >> void main() {  
2      var a = 1;  
3      methode(a, 2);  
4  }  
5  
6  void methode(int x, int y) {  
7      print(x * y);  
8  }
```

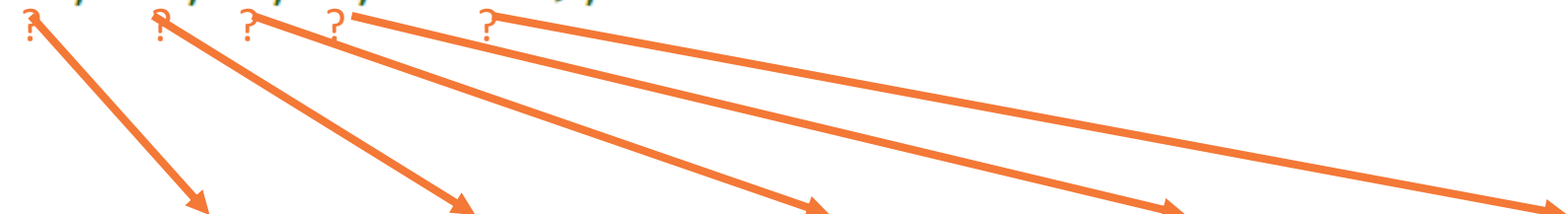
Two orange arrows originate from the arguments 'a' and '2' in the 'methode(a, 2);' call on line 3. One arrow points to the parameter 'x' in the 'methode' function definition on line 6, and the other points to the parameter 'y'.

Bitte nur zuhören
Nicht ausprobieren

Dart - Named Arguments

- Problem: man weiß nicht, was die Zahlen bedeuten

```
1  >> void main() {  
2      preis(300, 19, 0, 2, 19.90);  
3  }  
  
635 void preis(int netto, int steuer, int mengenrabbatt, int skonto, double versand) {  
636     print(netto *  
637         (1 + steuer / 100) *  
638         (1 - mengenrabbatt / 100) *  
639         (1 - skonto / 100) +  
640         versand);  
641 }
```



The diagram illustrates the mapping of arguments from the `main` function to the `preis` function. Five orange arrows originate from the arguments in the `preis(300, 19, 0, 2, 19.90);` call in the `main` function and point to the corresponding parameters in the `preis` function signature: `netto`, `steuer`, `mengenrabbatt`, `skonto`, and `versand`. Each arrow has a small red question mark at its starting point, highlighting the lack of semantic information provided by positional arguments.

Dart - Named Arguments

- Jetzt: Namensgebung beim Aufruf

```
1  >> void main() {  
2      var a = 1;  
3      methode(y←a, x←2);  
4  }  
5  
6  void methode({required int x, required int y) {  
7      print(x * y);  
8  }
```

Dart - Named Arguments

- Named arguments:

```
ergebnistyp name({ ... }) {  
    ...  
    return ergebnis;  
}
```

- Entweder required
required *typ argument*
- Oder mit Standardwert
typ argument2 = standardwert

Dart - Named Arguments

- Wie sieht es in unserem Problemfall aus?

Vorher:

```
1 >> void main() {  
2     preis(300, 19, 0, 2, 19.90);  
3 }
```



Nachher:

```
1 >> void main() {  
2     preis(netto: 300, steuer: 19, skonto: 2, versand: 19.90);  
3 }
```

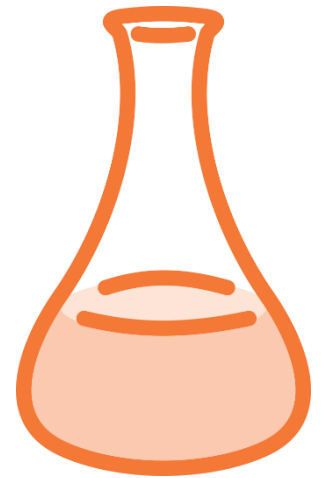


```
589 void preis({required int netto, required int steuer, int mengenrabbatt = 0,  
590             int skonto = 0, required double versand}) {  
591     print(netto * (1 + steuer / 100) * (1 - mengenrabbatt / 100) * (1 - skonto / 100) + versand);  
592 }
```

Dart - Named Arguments

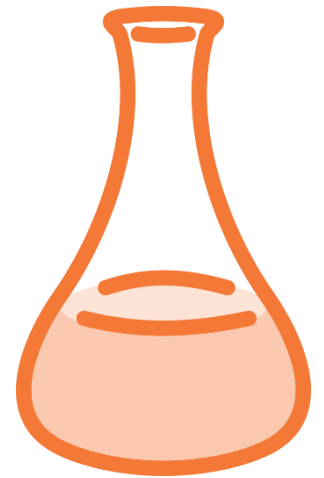
- Danke fürs Abwarten und Zuhören.
- Jetzt bitte ausprobieren:

```
1  >> void main() {  
2      preis(netto: 300, steuer: 19, skonto: 2, versand: 19.90);  
3  }  
  
589 void preis({required int netto, required int steuer, int mengenrabatt = 0,  
590             int skonto = 0, required double versand}) {  
591     print(netto * (1 + steuer / 100) * (1 - mengenrabatt / 100) * (1 - skonto / 100) + versand);  
592 }
```



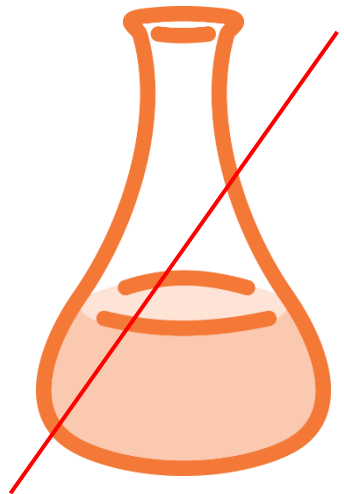
Clean Code

- Die Methode `preis()` tut zwei Dinge
 - Berechnung des Preises
 - Ausgabe auf dem Bildschirm
- Grundsatz: Methoden und Funktionen sollten nur 1 Aufgabe erledigen.
- Ändere die Methode zu einer Funktion, die nur rechnet und das Ergebnis liefert.
- Passe die `main()` Methode an, dass sie das Ergebnis auf dem Bildschirm ausgibt



Dart - Callbacks

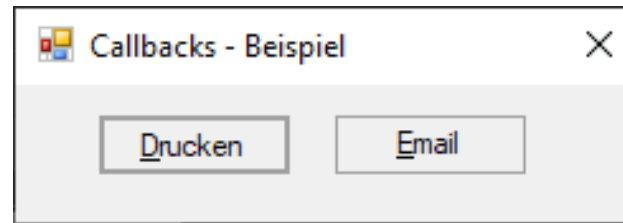
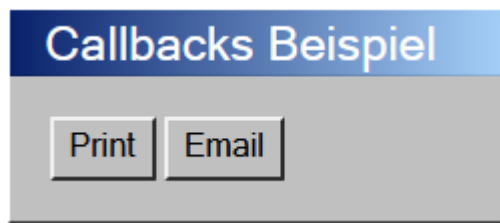
- Callbacks sind ein Mittel für die Erweiterbarkeit
- Motto:
 - Irgendwas soll passieren, aber ich kann im Moment nicht sagen, was genau.
 - Jemand anderes muss das entscheiden und mir dann mitteilen.



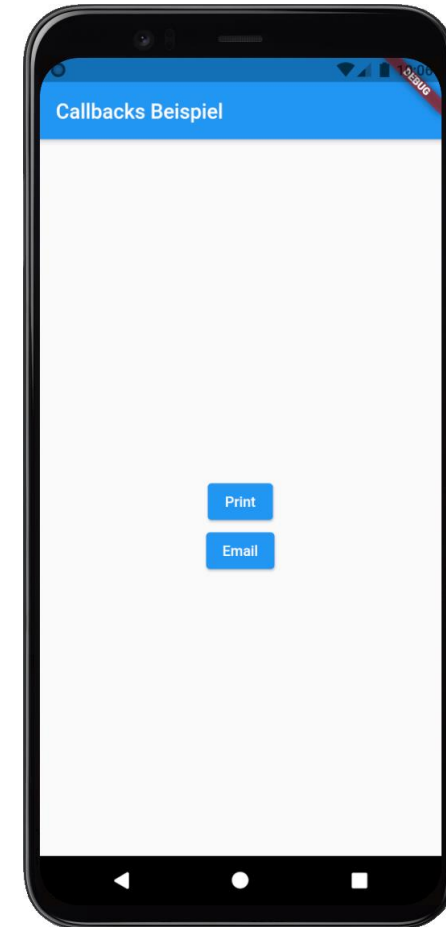
Bitte nur zuhören
Nicht ausprobieren

Dart - Callbacks

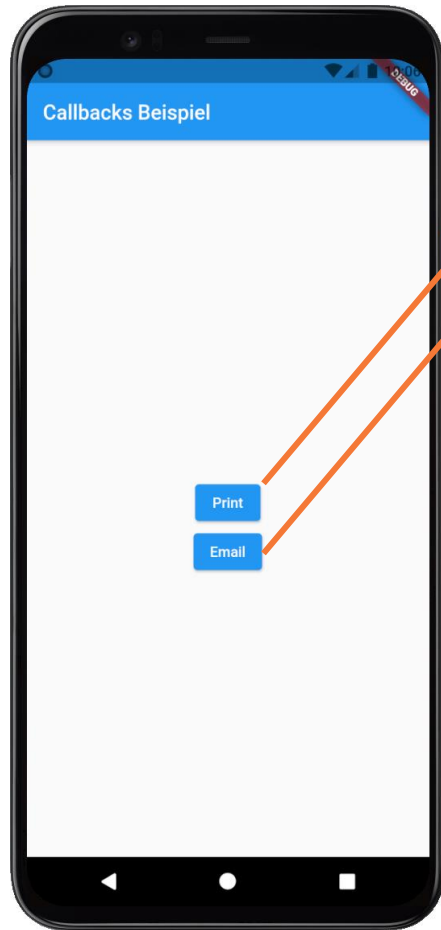
- Beispiel: Button



- Aussehen: soll jemand anders programmieren
- Verhalten: wollen wir bestimmen



Dart - Callbacks



```
ElevatedButton(onPressed: _print, child: Text("Print")),  
ElevatedButton(onPressed: _email, child: Text("Email")),
```

Callbacks

```
void _print() {  
  // TODO: Drucken  
}
```

```
void _email() {  
  // TODO: EMail senden  
}
```

Dart - Callbacks

- Mal vorausdenken:
Wie sieht die Funktion `preis()` in 3 Jahren aus?

```
589 void preis({required int netto, required int steuer, int mengenrabatt = 0,  
590     int skonto = 0, required double versand}) {  
591     print(netto * (1 + steuer / 100) * (1 - mengenrabatt / 100) * (1 - skonto / 100) + versand);  
592 }
```

```
5 void preis({  
6     required int netto,  
7     required int steuer,  
8     int mengenrabatt = 0,  
9     int skonto = 0,  
10    required double versand,  
11    double verpackung = 0,  
12    int grosskundenrabatt = 0,  
13    int promotionaktion = 0,  
14    double mindermengenzuschlag = 0,  
15    double sonderkonditionenVolkswagen = 0,  
16    double sonderspezialkonditionenPorsche = 0,  
17 }) {  
    ...
```

Dart - Callbacks

- Lösung: Callback

"Ruf mich an, wenn Du wissen willst, wie die Konditionen sind."

```
1  >> void main() {  
2      preis(netto: 300, steuer: 19, rabatt: keinRabatt, versand: 19.90);  
3  }  
4  
5  double keinRabatt(double betrag) {  
6      return betrag;  
7  }  
8  
9  void preis(  
10     {required double netto,  
11     required int steuer,  
12     required double Function(double) rabatt,  
13     required double versand}) {  
14     var rabattpreis = rabatt(netto);  
15     var bruttopreis = rabattpreis * (1 + steuer / 100);  
16     var versandpreis = bruttopreis + versand;  
17     print(versandpreis);  
18 }
```

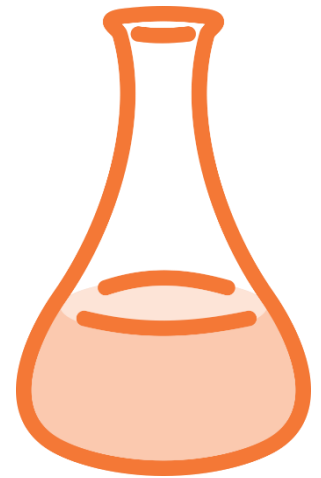
Argument

Aufruf der Callback-Funktion: wie sind die Konditionen?

Dart - Callbacks

- Programmierer zunächst nach:

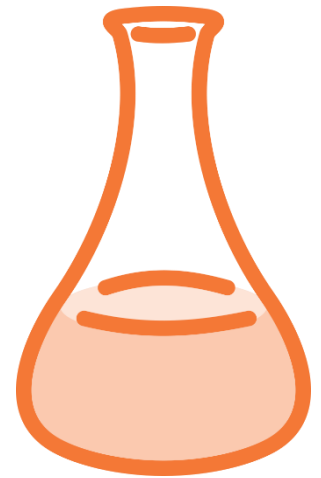
```
1  >> void main() {  
2      var p = preis(netto: 300, steuer: 19, rabatt: keinRabatt, versand: 19.90);  
3      print(p);  
4  }  
5  
6  double keinRabatt(double betrag) {  
7      return betrag;  
8  }  
9  
10 double preis(  
11     {required double netto,  
12     required int steuer,  
13     required double Function(double) rabatt,  
14     required double versand}) {  
15     var rabattpreis = rabatt(netto);  
16     var bruttopreis = rabattpreis * (1 + steuer / 100);  
17     var versandpreis = bruttopreis + versand;  
18     return versandpreis;  
19 }
```



Dart - Callbacks

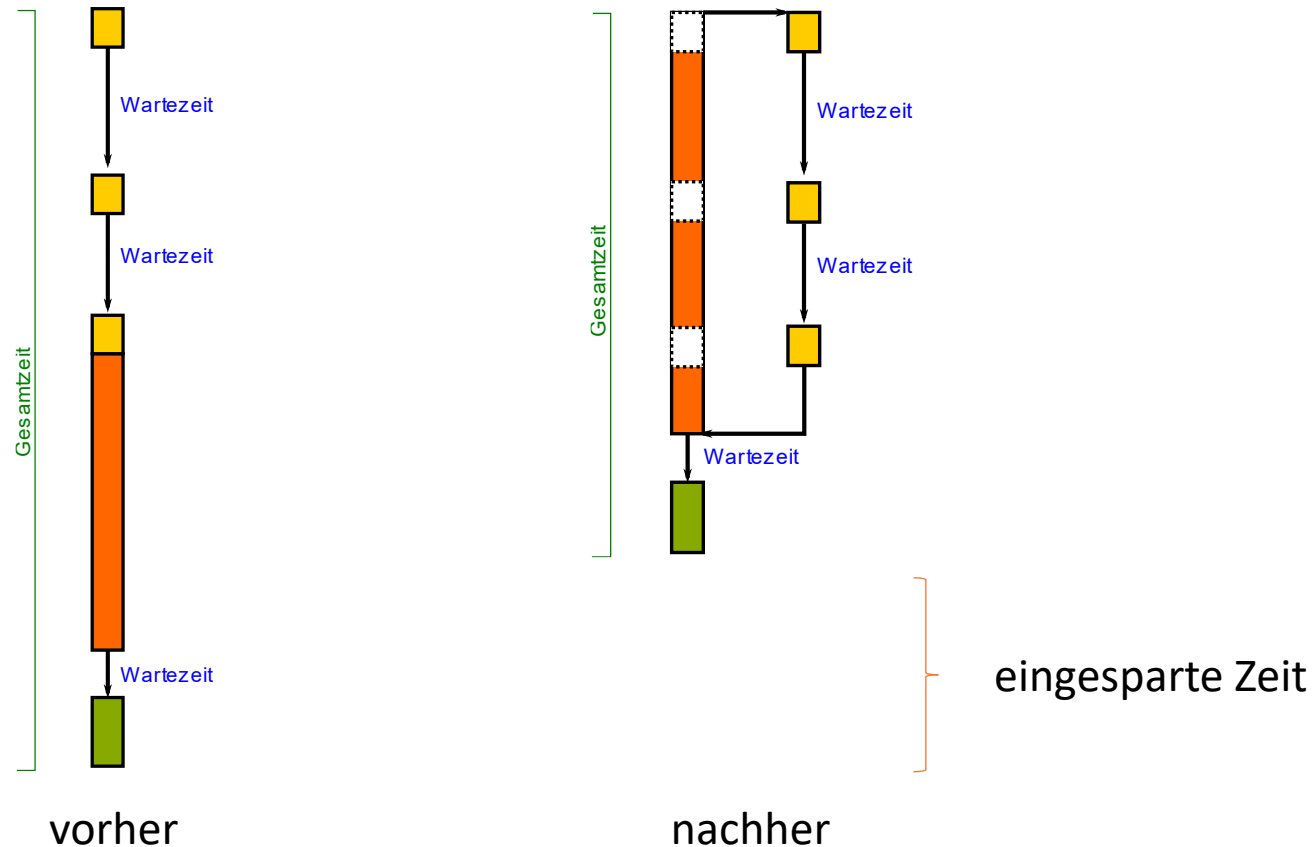
- Erweitere:
 - Eine Funktion für 2% Skonto
 - Eine Rabattfunktion für VW
 - Grundsätzlich 4% Rabatt
 - Ab 10000 € 6% Rabatt
- Folgender Code sollte laufen:

```
1  >> void main() {  
2      print(preis(netto: 300, steuer: 19, rabatt: keinRabatt, versand: 19.90));  
3      print(preis(netto: 300, steuer: 19, rabatt: skonto, versand: 19.90));  
4      print(preis(netto: 300, steuer: 19, rabatt: vw, versand: 19.90));  
5  }
```



Async/await

- Aufgaben im Hintergrund parallel erledigen lassen



Async/await

- Eine Funktion, die mit Pausen im Hintergrund ausgeführt werden kann, ist mit **async** versehen.
- Um von einer **async** Funktion ein Ergebnis zu bekommen, muss auf das Ergebnis mit **await** gewartet werden.
- "Normale" Funktionen: **var ergebnis = funktion();**
- Hintergrund-Funktionen: **var ergebnis = await funktion();**

Async/await

```
1 // Das programmieren wir selbst so
2 int normal() {
3     return 5;
4 }
5
6 // Das programmieren wir nicht selbst
7 // Es kann aber sein, dass wir es aus einer Bibliothek bekommen
8 Future<int> hintergrund() async {
9     return Future(() {
10         return 7;
11     });
12 }
13
14 void main() async {
15     var a = normal(); // so rufen wir eigene Funktionen auf
16     var b = await hintergrund(); // so rufen wir fremde Hintergrund-Methoden auf
17     print(a);
18     print(b);
19 }
```

Kennt ihr schon

Kommt aus einer Bibliothek

Objekte

- Paradigma Objektorientierung (OO)
 - Philosophie: "Alles ist ein Ding"
- Objekte sind "Gegenstände" mit konkreten Eigenschaften
 - Beispiel:
der Tisch mit 4 Beinen und hölzerner Tischplatte,
der bei Fritz im Büro steht,
am 12.5.2015 eingekauft wurde,
die Bestellnummer EAM 90061554 hat
und an der hinteren linken Ecke beschädigt ist
- Objekte werden auch *Instanzen* genannt

Objekte

- Eindeutig identifizierbar / Unikat
 - es gibt genau eins
 - ein genau gleich aussehendes Objekt ist trotzdem ein anderes
- dasselbe Objekt
 - identisches Unikat
 - besteht aus denselben Atomen
- ein gleiches Objekt
 - zwei identisch aussehende Objekte
 - besteht aus anderen Atomen
 - hat eine andere Position im Raum

Objekte

- Eindeutig identifizierbar / Unikat
 - es gibt genau eins
 - ein genau gleich aussehendes Objekt ist trotzdem ein anderes
- dasselbe Objekt
 - identisches Unikat
 - besteht aus denselben Atomen
- ein gleiches Objekt
 - zwei identisch aussehende Objekte
 - besteht aus anderen Atomen
 - hat eine andere Position im Raum

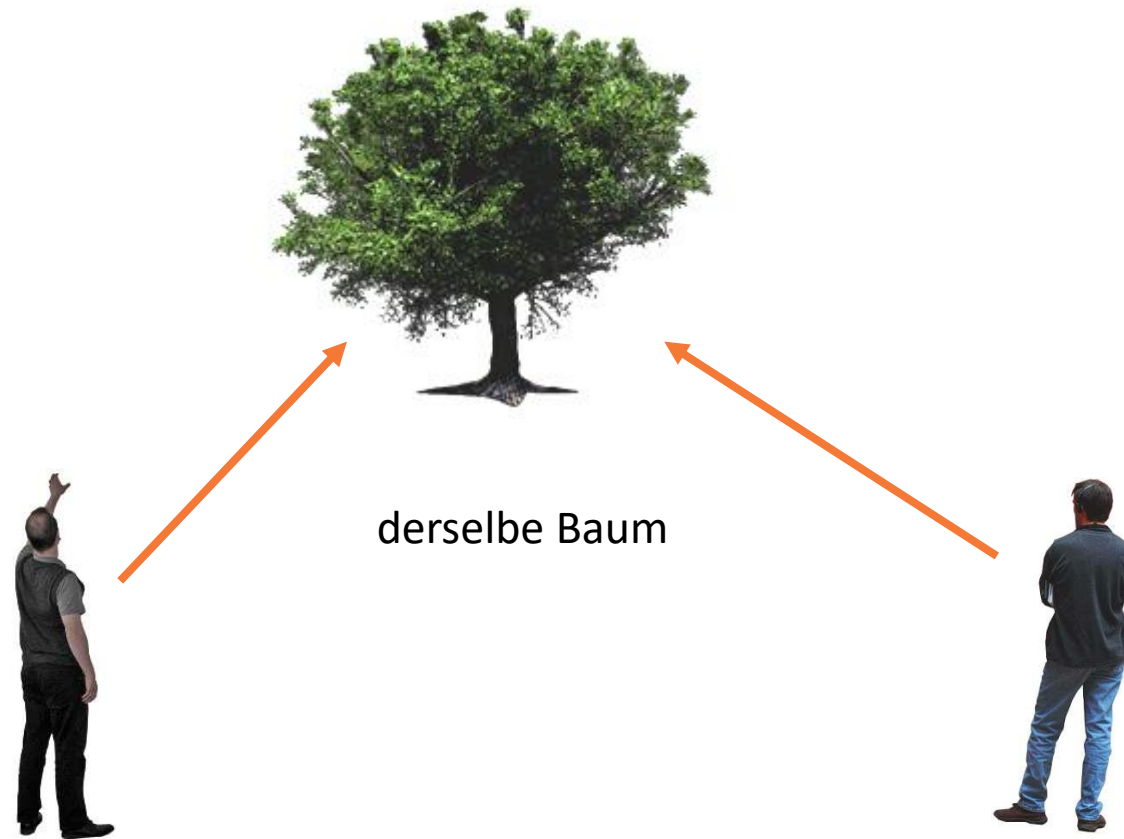
Elektronen

Elektronen

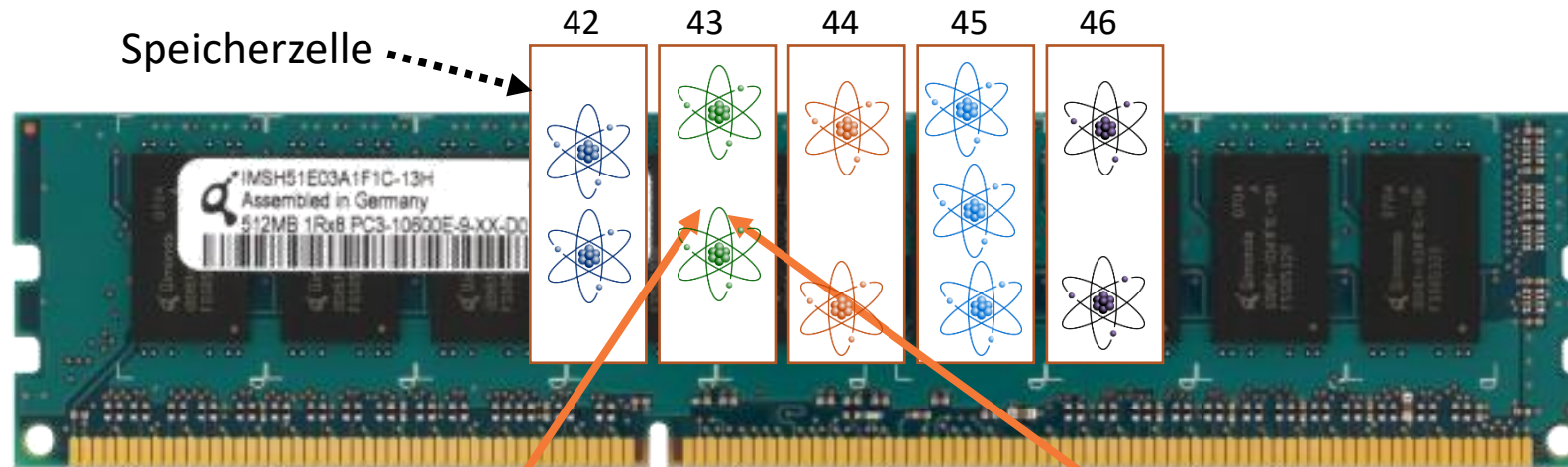
Adresse

RAM

Objekte



Objekte



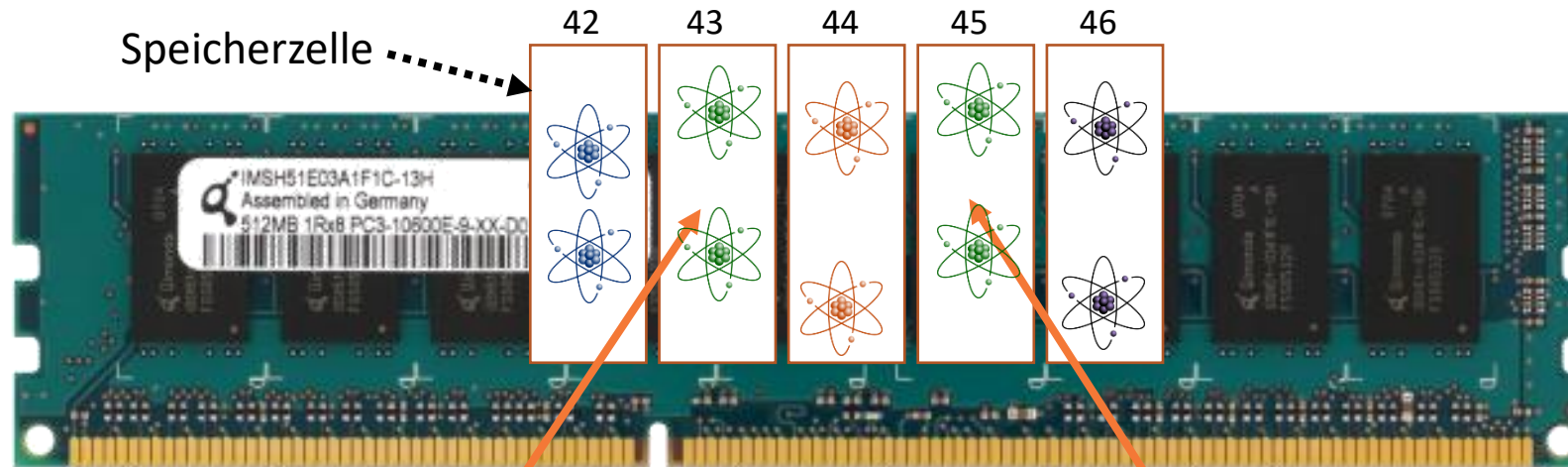
```
int calculate(int para, {required int named, int optional = 0}) {  
  return para + named + optional + 1;  
}  
  
void main() {  
  int result = Function.apply(calculate, [2], {Symbol("bar"): 4});  
  assert(result == 2 + 4 + 0 + 1);  
  
  result = Function.apply(calculate, [2], {#bar: 4});  
  assert(result == 2 + 4 + 0 + 1);  
  
  Symbol m = #main;  
  print(m);  
}
```

```
void main() {  
  var data = [12, 22, 910, 21.3];  
  for (var i = 0; i < data.length; ++i) {  
    var o = data[i];  
    var text = o.toString();  
    for (int j = 0; j < text.length; j++) {  
      var c = text.substring(0,2);  
      print(c);  
    }  
  }  
}
```

Objekte



Objekte



```
int calculate(int para, {required int named, int optional = 0}) {  
  return para + named + optional + 1;  
}  
  
void main() {  
  int result = Function.apply(calculate, [2], {Symbol("bar"): 4});  
  assert(result == 2 + 4 + 0 + 1);  
  
  result = Function.apply(calculate, [2], {#bar: 4});  
  assert(result == 2 + 4 + 0 + 1);  
  
  Symbol m = #main;  
  print(m);  
}
```

```
void main() {  
  var data = [12, 22, 910, 21.3];  
  for (var i = 0; i < data.length; ++i) {  
    var o = data[i];  
    var text = o.toString();  
    for (int j = 0; j < text.length; j++) {  
      var c = text.substring(0,2);  
      print(c);  
    }  
  }  
}
```

Objekte

- Warum ist das so wichtig?
 - Vorhersage, wie sich Änderungen auswirken



Klassen

- Klassen definieren die Eigenschaften, die Objekte haben können

- Beispiel:

Objekt:

"der Tisch mit 4 Beinen und hölzerner Tischplatte [...]"

Klasse:

ein Tisch hat eine Menge an Beinen

ein Tisch hat eine Tischplatte

eine Tischplatte besteht aus einem Material

[...]

Klassen

- Man kann aus Klassen Instanzen (Objekte) erzeugen
 - Beispiel:
Legoauto-Bauplan = Klasse
Hände, Finger, Werkzeug = Programmcode
Aufgebautes Spielzeug = Objekt
- Man sagt, ein Objekt sei vom Typ seiner Klasse
 - "Dieses Spielzeug ist vom Typ Legoauto"

Klassen

- Klassen können Methoden/Prozeduren definieren, um mit Objekten etwas zu tun
 - Beispiel:
Berechne Verkaufspreis (Tisch)
Drucke Liste benötigter Klötze (Legoauto)
Zeige CO₂ Auswirkung aufs Klima (Baum)

Klassen und Objekte in Dart

- Klassen werden mit `class Typname{...}` definiert

- Objekt erzeugen:

`var objekt = Typname();` ←.....

oder

`var objekt = new Typname();`

Bei dieser Schreibweise ist nicht direkt ersichtlich, ob es sich um eine Methode oder eine Klasse handelt. Es ist aber die modernere/neuere Schreibweise

Konvention:


Klassen = Substantiv (Hauptwort) → Großschreibung

Methoden = Verben (Tunwort) → Kleinschreibung

Klassen und Objekte in Dart

- Methoden oder Funktionen, die mit einem Objekt einer Klasse durchgeführt werden können:

```
class Typname {  
    void methode() {  
        // hier irgendwas tun  
    }  
}
```



Im Vergleich zu "normalen"
Methoden wird hier eingerückt

Klassen und Objekte in Dart

- Eigenschaften werden innerhalb der Klasse angegeben
- Sie können Werte bekommen in einer Methode, die gleich heißt wie die Klasse

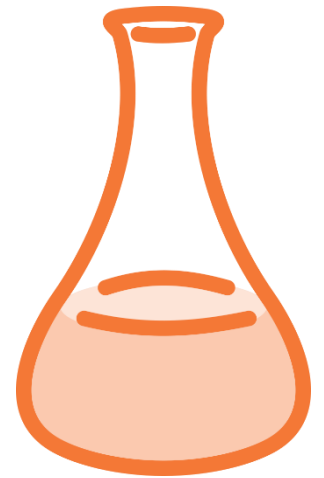
```
class Typname {  
    typ name = wert; // Eigenschaft, die gleich einen Wert hat  
    late typ name2; // Eigenschaft, die später einen Wert bekommt  
  
    Typname(typ argument) {  
        name2 = argument; // Jetzt hat sie einen Wert  
    }  
}
```

Klassen und Objekte in Dart

- Eigenschaften abfragen:
wert = objekt.eigenschaft;
- Operation (Methode) durchführen:
objekt.methodenname(argument1, ...);
- Berechnung (Funktion) ausführen:
ergebnis = objekt.methodenname(argument1, ...);

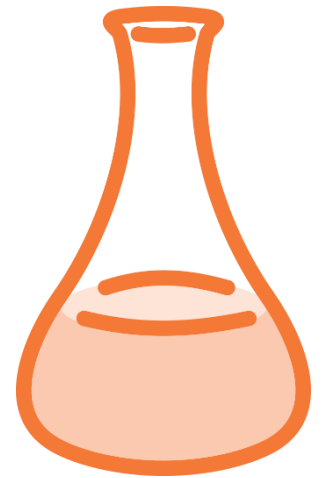
Klassen und Objekte in Dart

- Definiere eine Klasse Quader und gib ihr ein paar Eigenschaften, die ein Quader haben könnte
- Erzeuge einen Quader *a*
- Erzeuge einen anderen Quader *b*
- Erzeuge einen Quader *c*, der genau gleich aussieht wie Quader *b*
- Definiere einen Quader *d*, der identisch ist mit Quader *c*



Klassen und Objekte in Dart

- Ändere eine Eigenschaft an Quader ***b***
- Überprüfe die gleiche Eigenschaft an Quader ***c***
- Ändere eine Eigenschaft an Quader ***d***
- Überprüfe die Eigenschaft an Quader ***c***



Klassen und Objekte in Python

- Füge eine Methode zur Klasse hinzu, die den Quader um eine Achse um 90° dreht
- Füge eine Funktion zur Klasse hinzu, die das Volumen des Quaders ausrechnet



Zusammenfassung



- Dart ist eine kostenlose General Purpose Hochsprache
- Programmiert wird in reinem Text, die IDE übernimmt den Rest
- Variablen, Rechnen, Kommentare: `var x = 2+3*5; // sollte 17 sein`
- Texte: `var s = "x=${x}\n" + "_"*10;`
- Schleifen/Wiederholungen: `for` / `while`
- Wahrheitswerte und Logik: `var ergebnis = true and aussage;`
- Verzweigungen: `if` / `else if` / `else`
- Listen / Maps: `[...]` / `{... : ...}`
- Methoden / Funktionen: `typ name(typ arg, typ arg2) { return ergebnis; }`

Zusammenfassung



- Named Arguments:
`typ name({required typ arg, typ arg2 = standard}) { return ergebnis; }`
- Callback: `var callback = funktion`; (ohne Klammern)
- Async/await: `var ergebnis = await funktion()`;
- Objektorientierung: `class Typname { ... }` und `new Typname()`;

Fragen

