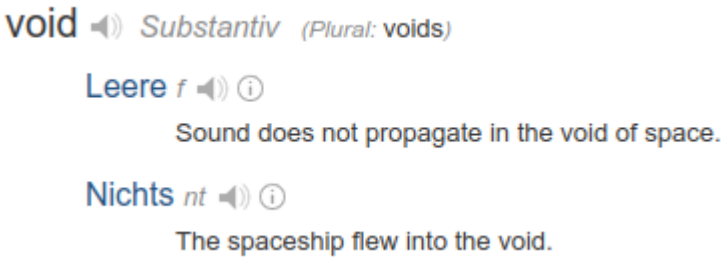
Funktionen - etwas ausführlicher

Im Lernmodul 1 zum Arduino hast du bereits Funktionen kennengelernt, um beispielsweise eine Abfolge von Tönen immer wieder abzuspielen, ohne jedes Mal alle Befehle für die Tonfolge erneut hinschreiben zu müssen. Hier soll es etwas ausführlicher um **Funktionen** und um die **Sichtbarkeit von Variablen** gehen.

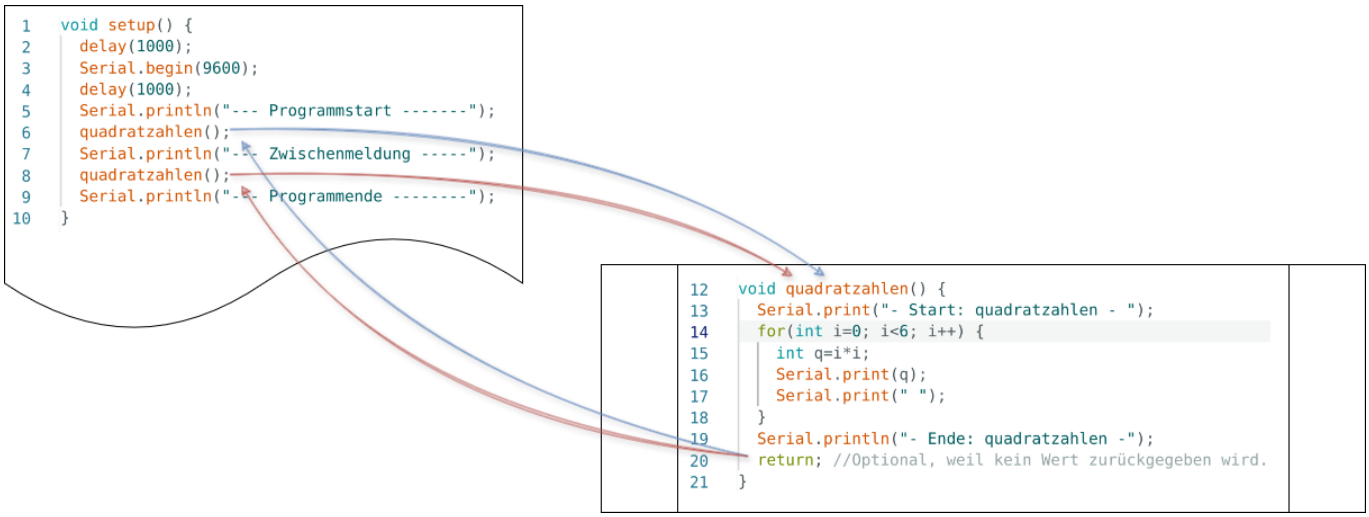
Funktionen mit einer Abfolge von Befehlen ohne Rückgabewert

Soll die Funktion – wo wie bei den Tonfolgen – nur eine Abfolge von Befehlen erledigen, muss sie mit dem Schlüsselwort `void` eingeleitet werden. Dieses Schlüsselwort gibt an, dass die Funktion „nichts“ zurückgibt.




```
void quadratzahlen() {
  Serial.print("- Start: quadratzahlen - ");
  for(int i=0; i<6; i++) {
    int q=i*i;
    Serial.print(q);
    Serial.print(" ");
  }
  Serial.println("- Ende: quadratzahlen -");
  return; //Optional, weil kein Wert zurückgegeben wird.
}
```

Wenn der Programmablauf zum Funktionsaufruf gelangt, springt die Verarbeitung zur Funktion, arbeitet die darin enthaltenden Anweisungen ab bis die return-Anweisung dazu führt, an die Stelle des Programms zurückzukehren, von wo der Funktionsaufruf gestartet wurde:



Bei Funktionen, die keinen Rückgabewert haben und bei denen erst ganz am Ende aller Anweisungen in der Funktion zum Aufrufpunkt zurückgekehrt werden soll, kann man die return-Anweisung weglassen.

(A1)

- Welche Ausgabe erscheint beim Ausführen des Sketches in der Abbildung auf der seriellen Konsole? Der `loop()` des Sketches ist leer.
- Erstelle einen Sketch mit der Funktion `quadratzahlen` und den im Bild sichtbaren Aufrufen im Setup. Überprüfe, ob deine Antwort richtig war, indem du den Sketch ausführst und die Ausgabe kontrollierst.
- Wieverändert sich die Ausgabe, wenn man den Code der Funktion wie unten gezeigt verändert? Überlege erst und teste dann.

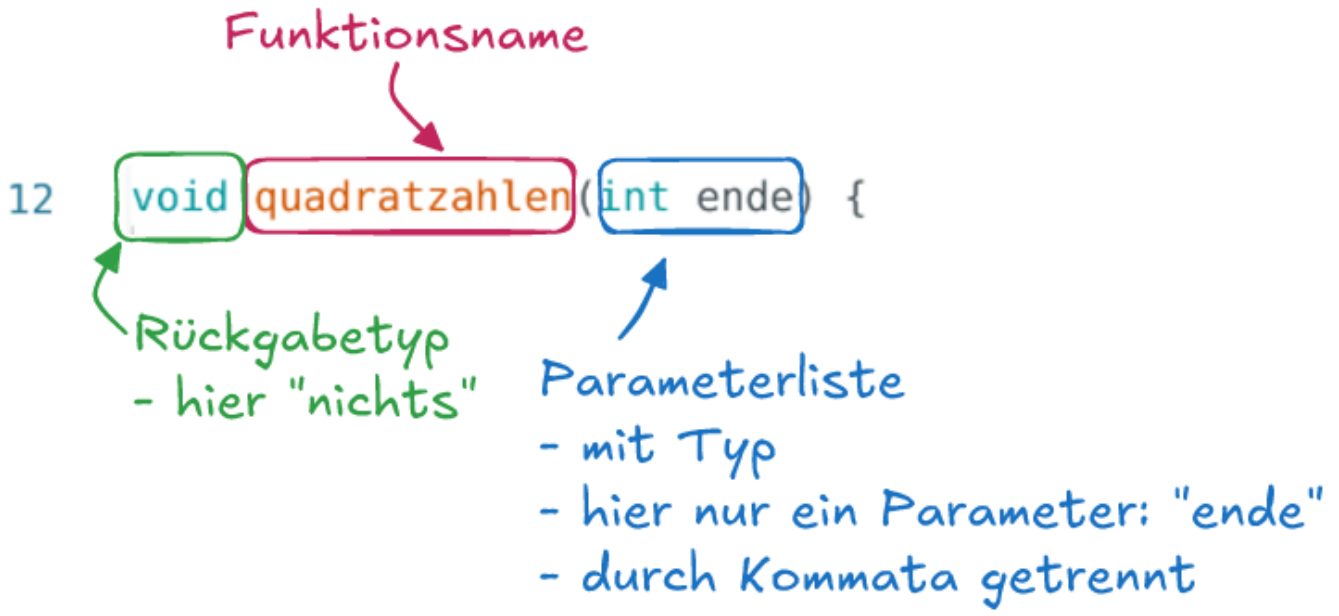
```
void quadratzahlen() {
  Serial.print("- Start: quadratzahlen - ");
  for(int i=0; i<6; i++) {
    int q=i*i;
    Serial.print(q);
    Serial.print(" ");
  }
  return;
  Serial.println("- Ende: quadratzahlen -");
  return; //Optional, weil kein Wert zurückgegeben wird.
}
```

Funktionen mit Parametern, aber ohne Rückgabewert

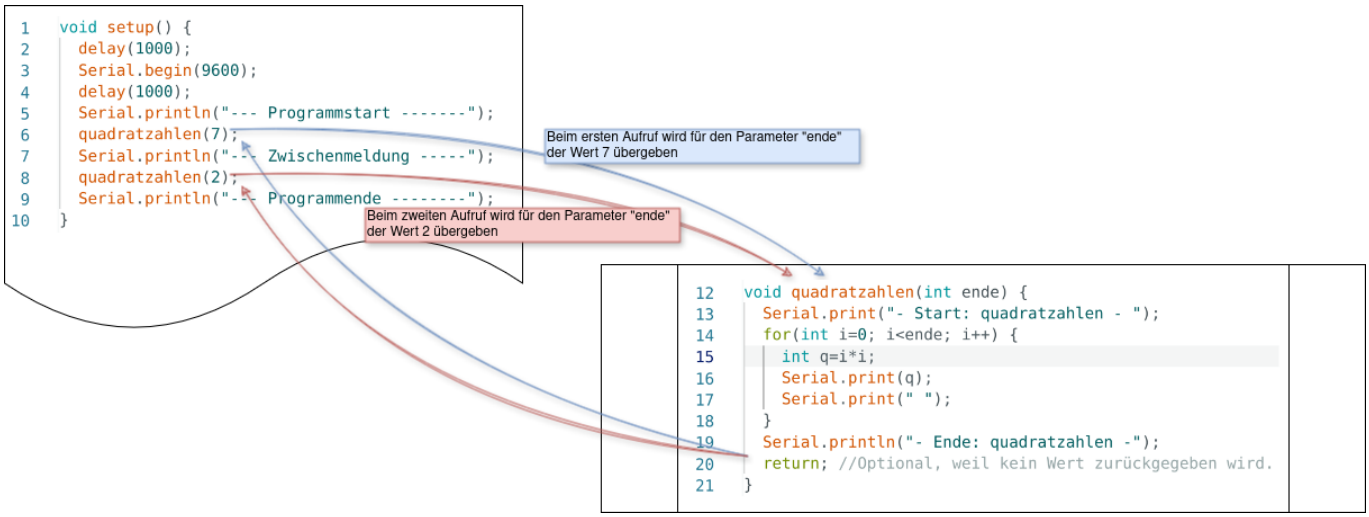
Um Funktionen flexibler einsetzen zu können, indem man ihr Verhalten steuert, kann man ihnen einen oder mehrere **Parameter** übergeben:

```
void quadratzahlen(int ende) {
  Serial.print("- Start: quadratzahlen - ");
  for(int i=0; i<ende; i++) {
    int q=i*i;
    Serial.print(q);
    Serial.print(" ");
  }
  Serial.println("- Ende: quadratzahlen -");
  return; //Optional, weil kein Wert zurückgegeben wird.
}
```

Die „Signatur“ der Funktion zeigt uns jezt an, dass sie nichts zurückgibt („void“) aber einen Parameter vom Typ `int` erwartet, wenn sie aufgerufen wird:



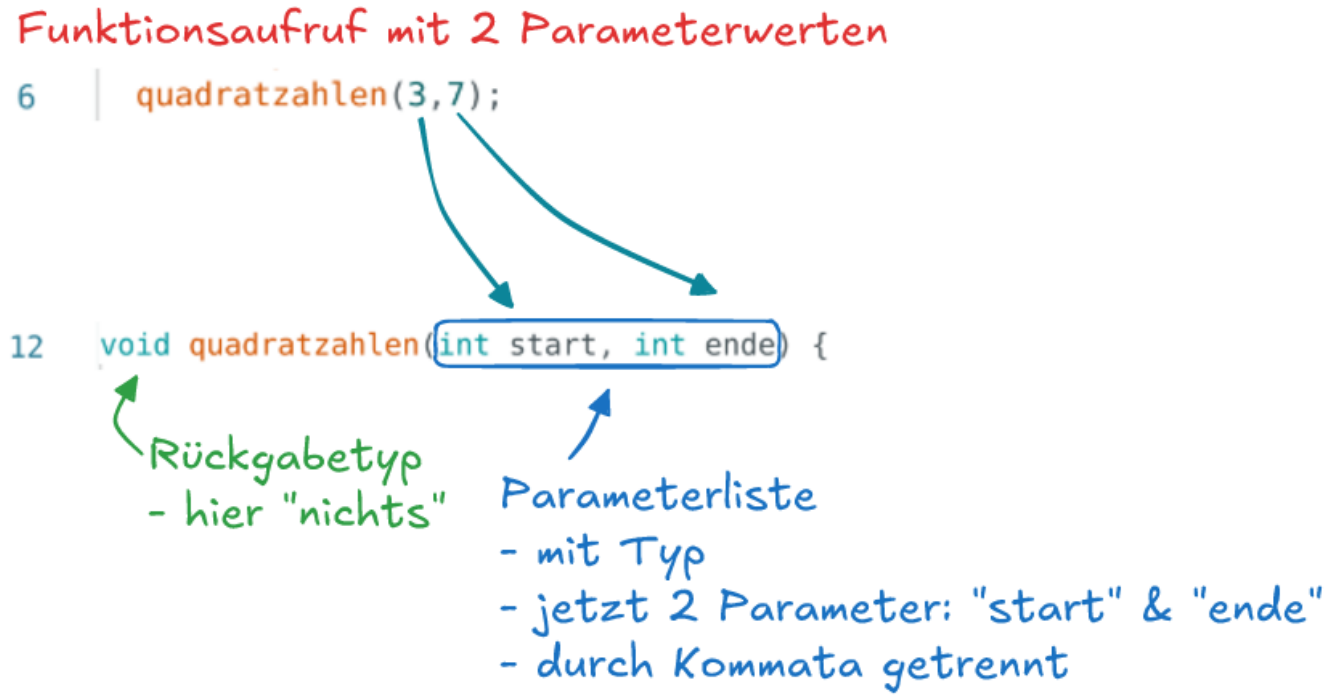
Jeder Parameter, der an die Funktion übergeben wird, erzeugt eine **lokale Variable**, die nur innerhalb der geschweiften Klammern der Funktion „sichtbar“ ist. Die Variable `ende` existiert also nur, solange die Funktion `quadratzahlen` verarbeitet wird und hat dabei jeweils den Wert, den sie beim Aufruf der Funktion von der Aufrufenden Stelle erhalten hat.



(A2)

- Welche Ausgabe erscheint jetzt beim Ausführen des Sketches in der Abbildung auf der seriellen Konsole? Der `loop()` des Sketches ist leer.
- Ändere deinen Sketch ab und teste.
- Probiere weitere Parameterwerte aus.

Man kann auch mehr als einen Parameter an eine Funktion übergeben:



Beim Aufruf der Funktion müssen die Zahl und dieTypen den Parameter mit den in den Signatur festgelegten Rahmenbedingungen übereinstimmen, sonst erhältst du bereits beim Übersetzen des Programms eine Fehlermeldung.

(A3)

- Ändere deinen Sketch so ab, dass man beim Aufruf der Funktion festlegen kann, welche Quadratzahlen ausgegeben werden, von `start` bis ende.
- Was passiert, wenn du die Funktion jetzt so aufrufst:
  - `quadratzahlen(6)`
  - `quadratzahlen(2.1,5)`
  - `quadratzahlen(„a“,5)`
- Kannst du einen dritten Parameter für die Schrittweite einführen, so dass du z.B. nur die geraden Quadratzahlen erhältst?

Lokale und globale Variablen

Im folgenden Sketch habe ich die Parameter wieder entfernt, die Ausgabe der Quadratzahlen erfolgt

dennoch nur von 1 bis 25:

```
// Globale Variablen
int start=1;
int ende=5;

void setup() {
  delay(1000);
  Serial.begin(9600);
  delay(1000);
  Serial.println("--- Programmstart -------");

  quadratzahlen();
  Serial.println("--- Zwischenmeldung -----");
  quadratzahlen();
  Serial.println("--- Programmende --------");
}

void quadratzahlen() {
  Serial.print("- Start: quadratzahlen - ");
  for(int i=start; i<ende; i++) {
    int q=i*i;
    Serial.print(q);
    Serial.print(" ");
  }
  Serial.println("- Ende: quadratzahlen -");
  return; //Optional, weil kein Wert zurückgegeben wird.
}

void loop() {
// Leer - der Loop hat hier keine Funktion
}
```

(A4)

- Erkläre, warum sich das Programm so verhält.
- Übernehme die Änderungen in deinen Sketch und teste.
- Ändere den Wert von `start` im Setup auf 3.
- Kommentiere die beiden Variablendeklarationen am Beginn des Sketsches aus und teste, was jetzt passiert.

> Variablen, die am Beginn eines Arduino Sketches deklariert werden, sind **global** gültig: Man kann in allen Funktionen auf sie zugreifen.
>
> Allerdings kann man sie auch überall verändern. Bei längeren Programmen ist es darum oft

nicht immer einfach zu wissen, was für einen Wert eine globale Variable gerade hat.

Man sollte die Verwendung von globalen Variablen darum auf wenige Zwecke einschränken, z.B. um den PINs des Arduino „sprechende Namen“ zu geben.

Man kann den Sketch wie folgt abändern, um lokale Variablen zu erhalten:

```
void setup() {
  delay(1000);
  Serial.begin(9600);
  delay(1000);
  Serial.println("--- Programmstart -------");
  quadratzahlen();
  Serial.println("--- Zwischenmeldung -----");
  quadratzahlen();
  Serial.println("--- Programmende --------");
}

void quadratzahlen() {
// Lokale Variablen
int start=1;
int ende=5;
  Serial.print("- Start: quadratzahlen - ");
  for(int i=start; i<ende; i++) {
    int q=i*i;
    Serial.print(q);
    Serial.print(" ");
  }
  Serial.println("- Ende: quadratzahlen -");
  return; //Optional, weil kein Wert zurückgegeben wird.
}
```

Variablen, die innerhalb eines Codeblocks mit geschweiften Klammern deklariert werden, gelten nur **lokal** innerhalb dieses Klammerblocks. Außerhalb des Klammerblocks kann nicht auf sie zugegriffen werden.

- Parameter einer Funktion sind immer lokale Variablen in der betreffenden Funktion.
- Laufvariablen von for-Schleifen sind lokal innerhalb des Schleifenblocks

(A5)

Ändere deinen Sketch so ab, dass lokale Variablen verwendet werden. Versuche die Variable `start` im Setup-Block auf den Wert 3 zu setzen.

Was ist, wenn es globale und lokale Variablen mit gleichem Namen gibt?

Betrachte den folgenden Sketch:

```
int start=2;
void setup() {
  delay(1000);
  Serial.begin(9600);
  delay(1000);
  Serial.println("--- Programmstart -------");
  quadratzahlen();
  Serial.println("--- Zwischenmeldung -----");
  quadratzahlen();
  Serial.println("--- Programmende --------");
}

void quadratzahlen() {
// Lokale Variablen
int start=1;
int ende=5;
  Serial.print("- Start: quadratzahlen - ");
  for(int i=start; i<ende; i++) {
    int q=i*i;
    Serial.print(q);
    Serial.print(" ");
  }
  Serial.println("- Ende: quadratzahlen -");
  return; //Optional, weil kein Wert zurückgegeben wird.
}
```

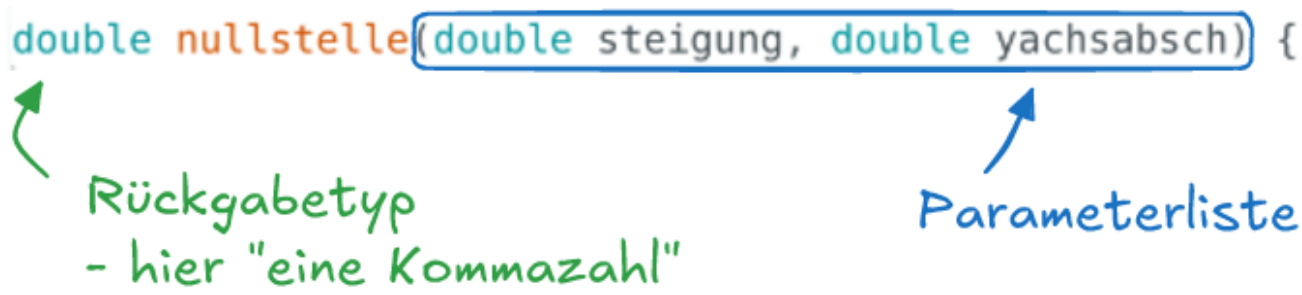
Hier gibt es sowohl eine globale als auch eine lokale Variable mit dem Namen `start`. Welche wird jetzt in der for-Schleife verwendet?

(A6)

- Teste den Sketch. Welche Variable wird in der for-Schleife verwendet?
- Gib den Wert von `start` irgendwo im setup mit einem Serial.printl Befehl aus - welcher Wert wird hier ausgegeben?

Funktionen mit Rückgabewert

Funktionen können, wenn sie zu der Stelle zurückkehren, an der sie aufgerufen wurden auch noch Ergebnisse zurückgeben.




```
void setup() {
  delay(1000);
  Serial.begin(9600);
  delay(1000);
  Serial.println("--- Programmstart -------");
  double n = nullstelle(-2,4);
  Serial.print("Nullstelle bei x=");
  Serial.println(n);
  Serial.println("--- Programmende --------");
}

double nullstelle(double steigung, double yachsabsch) {
  double nullst = -1 * yachsabsch / steigung;
  return nullst;
}

void loop() {
// Leer - der Loop hat hier keine Funktion
}
```

(A7)

- Was berechnet der abgebildete Sketch? Teste mit verschiedenen Parameterwerten.
- Erweitere den Aufruf der Funktion im Setup so, dass alle Nullstellen aller Geraden mit Steigungen zwischen -2.0 und -1.0 mit der Schrittweite 0.1 berechnet werden:

```
--- Programmstart -------
Steigung m=-2.00 Nullstelle bei x=2.00
Steigung m=-1.90 Nullstelle bei x=2.11
Steigung m=-1.80 Nullstelle bei x=2.22
Steigung m=-1.70 Nullstelle bei x=2.35
Steigung m=-1.60 Nullstelle bei x=2.50
Steigung m=-1.50 Nullstelle bei x=2.67
Steigung m=-1.40 Nullstelle bei x=2.86
Steigung m=-1.30 Nullstelle bei x=3.08
Steigung m=-1.20 Nullstelle bei x=3.33
Steigung m=-1.10 Nullstelle bei x=3.64
Steigung m=-1.00 Nullstelle bei x=4.00
--- Programmende --------
```

Hilfestellung 1: For Schleife mit 0.1er Schritten

Die for-Schleife sieht hier z.B. so aus:

```
for (double i = -2; i <= -0.99; i = i + 0.1) {
    // Rechnen und ausgeben
  }
```