



Timetable:

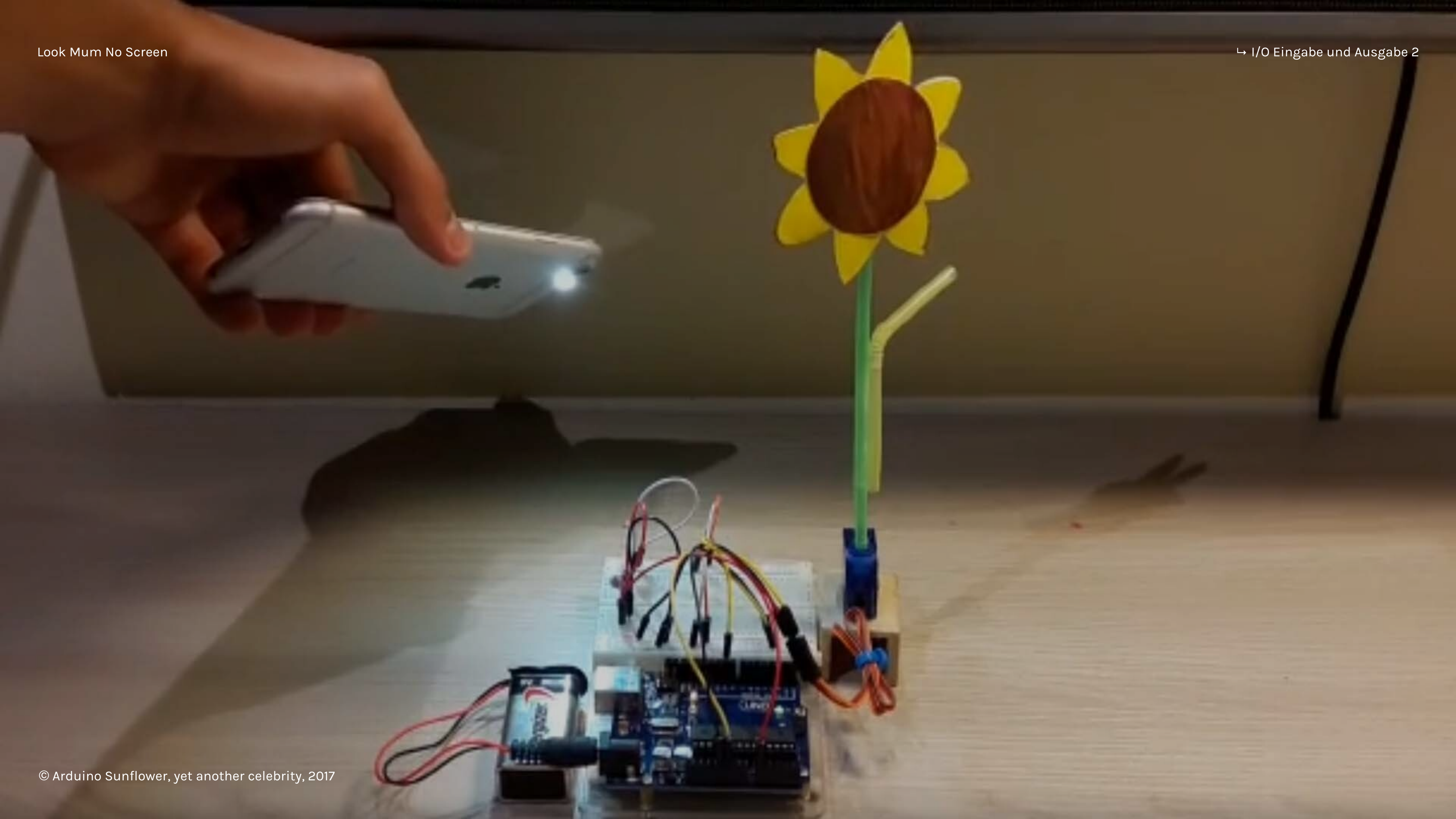
Kw 42	20.10.2021 (Kursvorstellung)
Kw 43	27.10.2021
Kw 44	03.11.2021 (Kick Off)
Kw 45	10.11.2021
Kw 46	17.11.2021 (I/O Eingabe und Ausgabe)
Kw 47	24.11.2021
Kw 48	01.12.2021 (I/O Eingabe und Ausgabe 2)
Kw 49	08.12.2021
Kw 50	15.12.2021 (Zwischenpräsentation)
Kw 51	22.12.2021
Kw 52	29.12.2021
Kw 01	05.01.2022
Kw 02	12.01.2022
Kw 03	19.01.2022
Kw 04	26.01.2022
Kw 05	02.02.2022 (white card)
Kw 06	09.02.2022 (Abschlusspräsentation)

Der Kurs **Look Mum No Screen** beschäftigt sich mit der Frage, wie Inhalte und Informationen aus dem Digitalen, in den analogen Raum transportiert werden können.

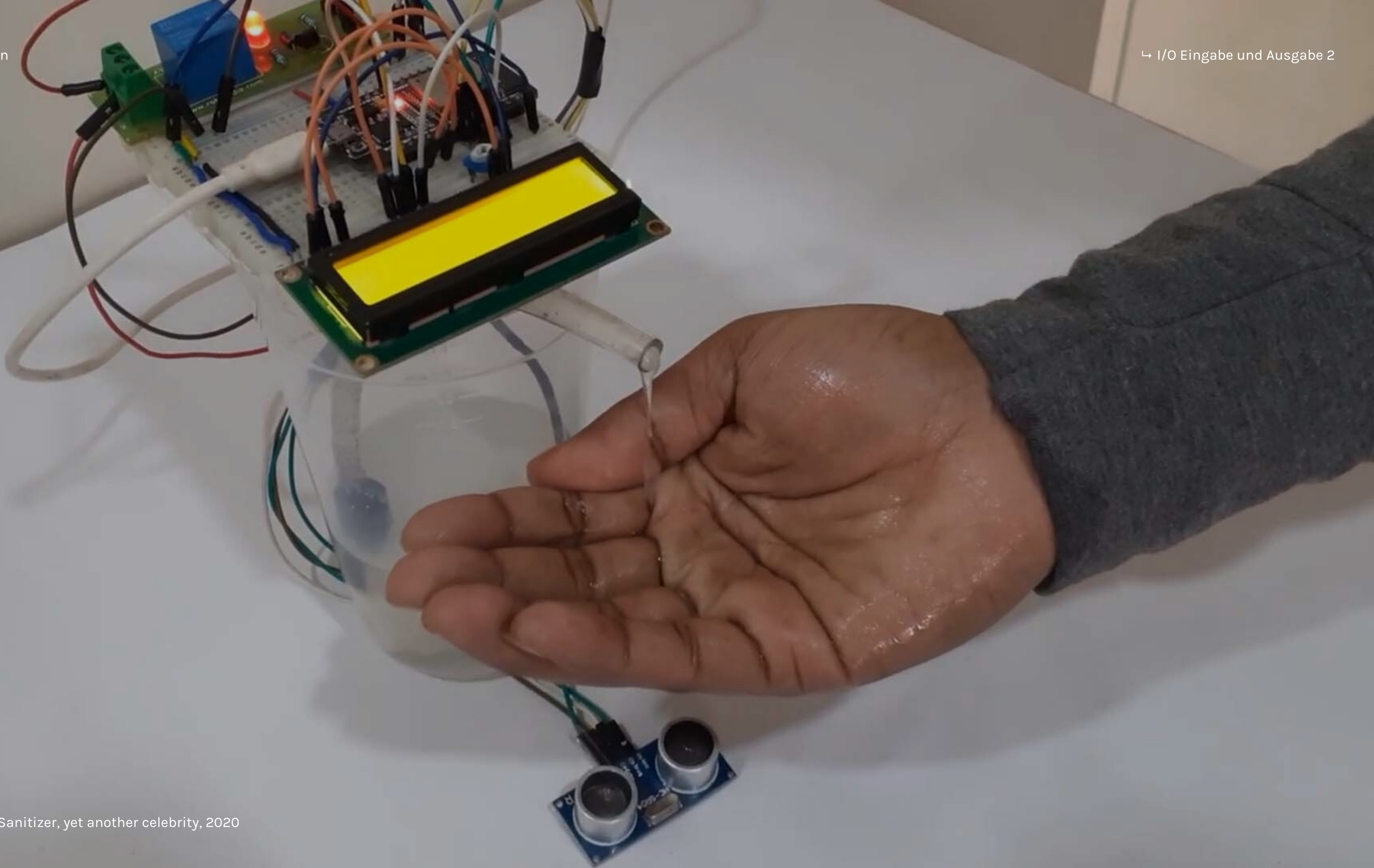
...

In der Veranstaltung **I/O Eingabe und Ausgabe 2** geben wir weiterhin Einblicke in die Welt des Physical-Computing. Wir präsentieren Arduino-Projekte und sprechen über die technische Umsetzung. Inhaltlich soll es um Funktionen und ihre Parameter, arithmetische Operatoren, weitere Datentypen sowie for- und while-Schleifen gehen. Alle inhaltlich angesprochenen Themen werden dabei mit Grafiken bebildert.





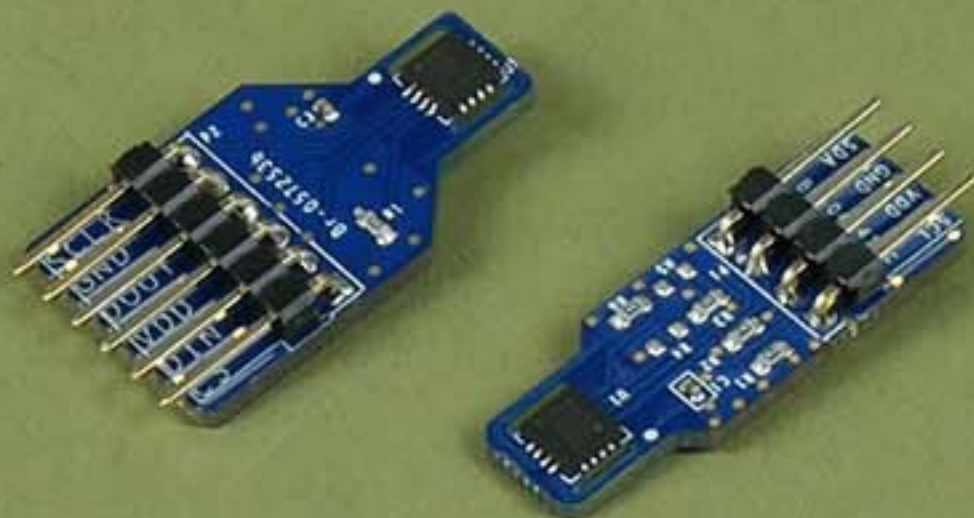






HC-SR04
Ultraschall
Abstandssensor:

Der Distanzrahmen liegt zwischen 3 bis 400 cm. Die maximale Abweichung beträgt hierbei gerade einmal 3 Millimeter. Durch die kleinen Abmessungen und hohe Verarbeitungsqualität ist dieses Modul auch ideal für mobile Einsätze geeignet.

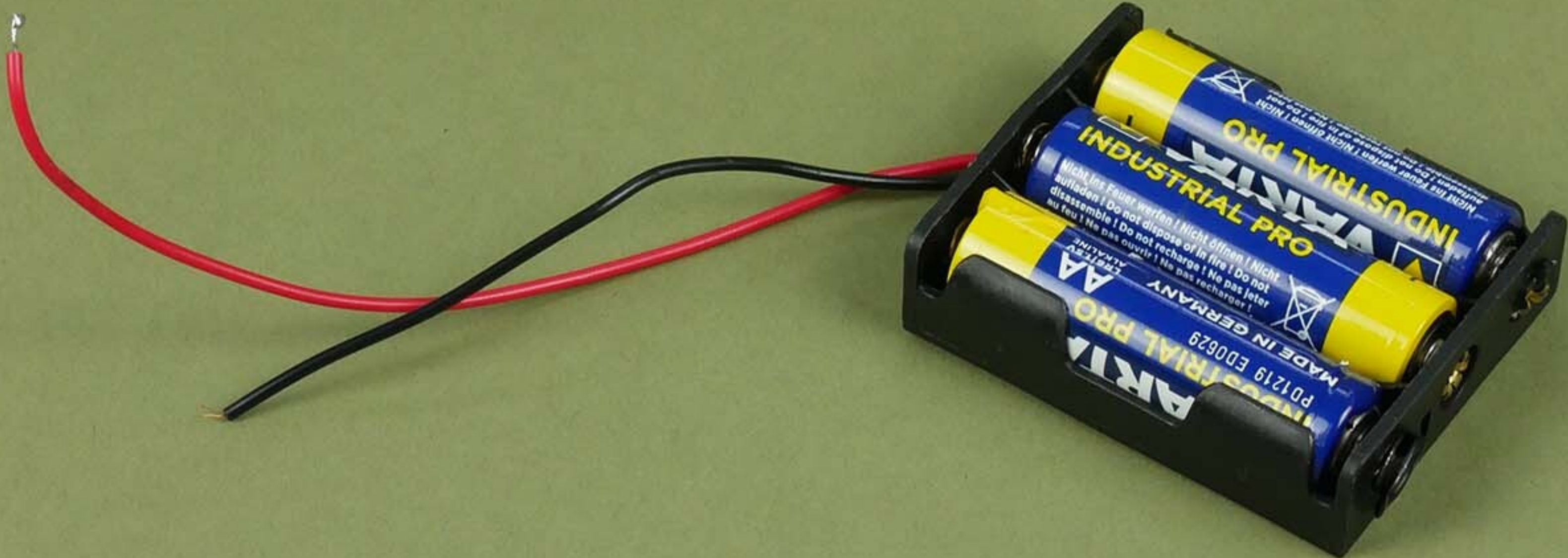




ARD SEN2
KNOCK:

Dieser Sensor registriert Schläge und gibt bei treffen ein **HIGH** Signal am Ausgang aus. Diese Module werden auch als Klopfswitch bezeichnet und können unter anderem zur Berührungserkennung eingesetzt werden.





Aufbau einer
Funktion:

Der Aufbau einer Funktion verfolgt ein bestimmtes Schema.
Eine Funktion macht dann Sinn, wenn es sich um eine
wiederholende Aufgaben bzw. Anweisung handelt.

```
void setup(){  
  // put your setup code here, to run once:  
  
}
```

```
void loop(){  
  // put your main code here, to run repeatedly:  
  
}
```

```
Typ funktionsName(parameter){  
  // schreibe deine Anweisungen hier:  
  
}
```

Aufbau einer
Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.


Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5;  
    int y = x + 2;  
    return y;  
}
```


Aufbau einer Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.

Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5;  // Variable x mit dem Wert 5  
    int y = x + 2;  
    return y;  
}
```

Aufbau einer Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.


Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5;  
    int y = x + 2; ← // Addition der Variable x + 2  
    return y;  
}
```


Aufbau einer Funktion:

Funktionen werden erstellt indem zuerst der Type der Funktion definiert wird. Dieser ist identisch mit dem Datentyp des zurückgegebenen Wertes.

Wenn kein Wert zurückgegeben werden soll, so wird der Funktionstyp **void** verwendet. Nach der Definition des Types wird der Name festgelegt und in Klammern alle Parameter, die der Funktion übergeben werden sollen.

```
void setup(){  
    // put your setup code here, to run once:  
  
}  
  
int calculate(){  
    int x = 5;  
    int y = x + 2;  
    return y;  // Was wird zurück gegeben?  
}
```

Aufbau einer
Funktion:

Geschweifte Klammern definieren den Anfang und das Ende von Funktions- und Anweisungsblöcken. Diese werden ebenfalls bei **for-Schleifen** und **if-Anweisung** verwendet.

```
int calculate(){ // geöffnete Klammer
    int x = 5;
    int y = x + 2;
    return y;
} // geschlossene Klammer
```


Aufruf einer
Funktion:

Unsere Funktion wird mit dem Funktionsnamen und den beiden
Klammern aufgerufen.

```
int calculate(){
    int x = 5;
    int y = x + 2;
    return y;
}

calculate(); ← // Funktion wird aufgerufen
```

Parameter einer
Funktion:

Um die Flexibilität unserer Funktion zu erhöhen, können wir ihr Funktionsparameter übergeben. Um die Parameter zu übergeben, müssen wir der Funktion zunächst mitteilen, welche Art von Parameter sie erwarten soll, und ihnen Variablennamen zuweisen.

```
int calculate(parameter_1, parameter_2){ ←—————  
    int x = 5;  
    int y = x + 2;  
    return y;  
}
```


Parameter einer
Funktion:


Um die Flexibilität unserer Funktion zu erhöhen, können wir ihr Funktionsparameter übergeben. Um die Parameter zu übergeben, müssen wir der Funktion zunächst mitteilen, welche Art von Parameter sie erwarten soll, und ihnen Variablennamen zuweisen.

```
int calculate(int x){ ←  
    int x = 5;  
    int y = x + 2;  
    return y;  
}
```

Parameter einer
Funktion:

Dies geschieht, indem wir die Parameter auf die gleiche Weise wie
im Code definieren, jetzt aber in den Klammern nach dem
Funktionsnamen.

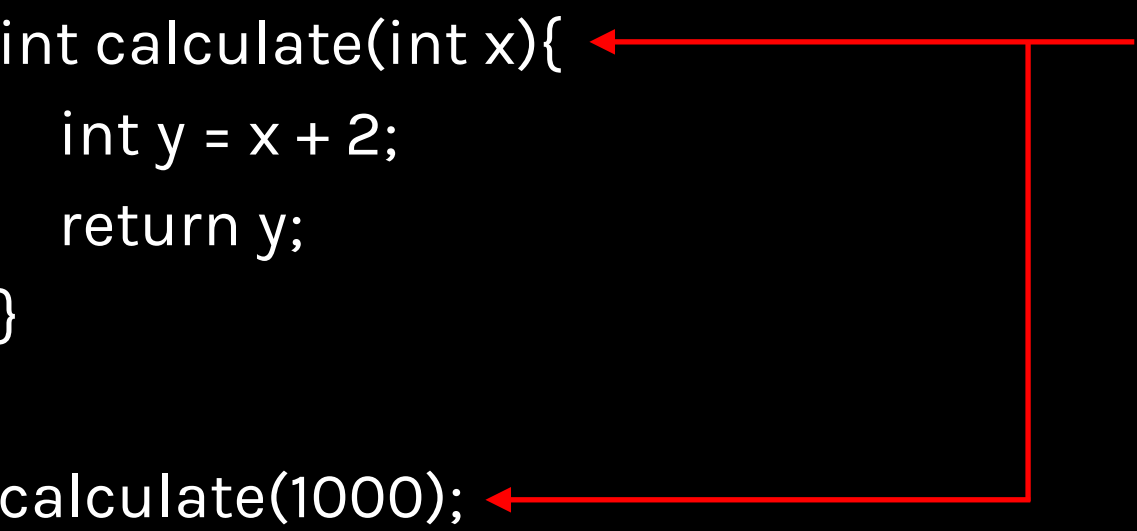
```
int calculate(int x){  
    int y = x + 2;  
    return y;  
}
```



Parameter einer
Funktion:

Dies geschieht, indem wir die Parameter auf die gleiche Weise wie
im Code definieren, jetzt aber in den Klammern nach dem
Funktionsnamen.

```
int calculate(int x){  
    int y = x + 2;  
    return y;  
}  
  
calculate(1000);
```



The diagram illustrates the process of passing an argument to a function. A red line originates from the value '1000' in the function call 'calculate(1000);' at the bottom. This line extends horizontally to the right, then turns 90 degrees upwards, and finally turns 90 degrees to the left, ending with an arrowhead pointing to the parameter 'x' in the function definition 'int calculate(int x){' at the top. This visualizes how the value 1000 is substituted for the variable x when the function is executed.

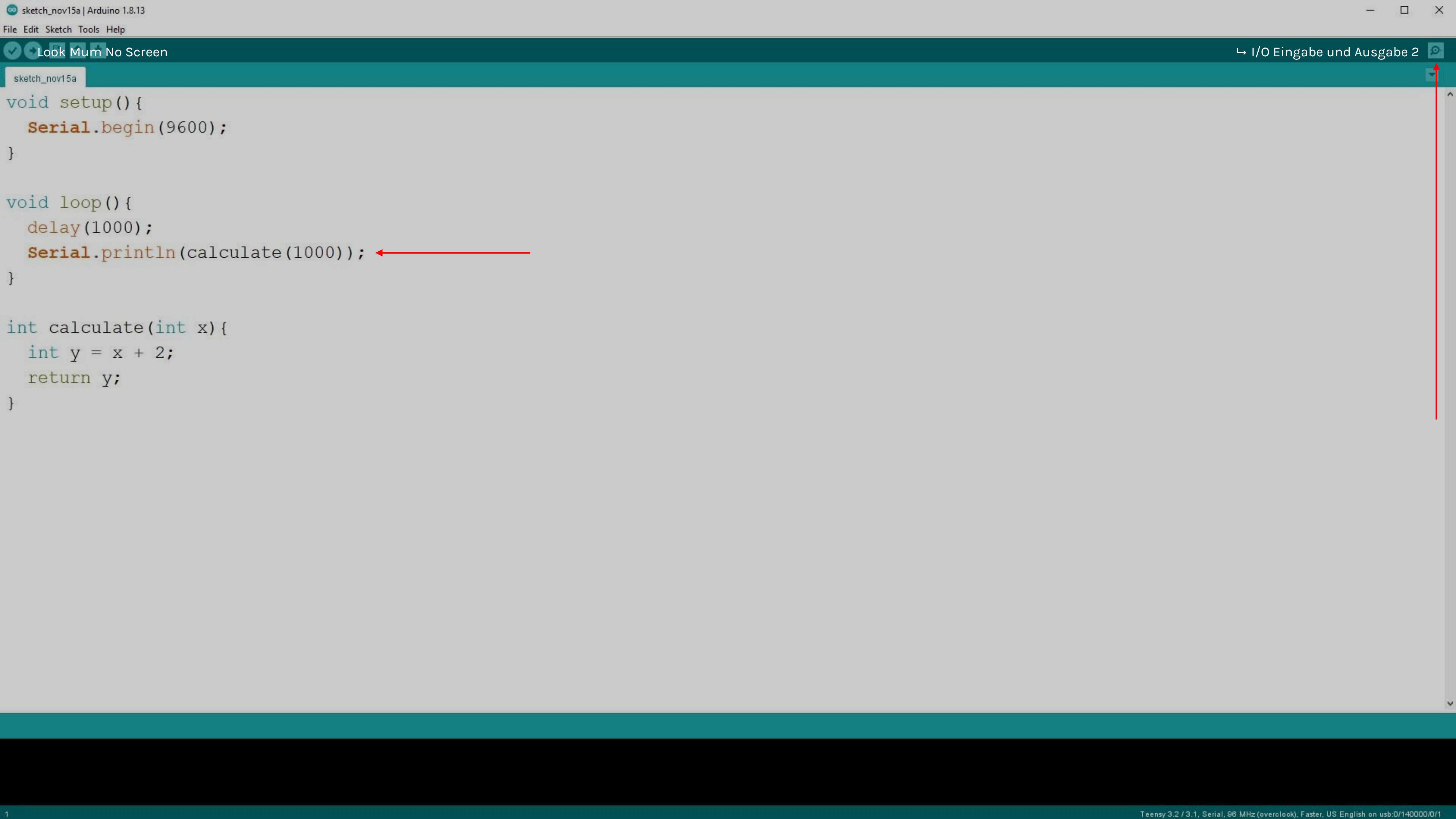
Parameter einer
Funktion:

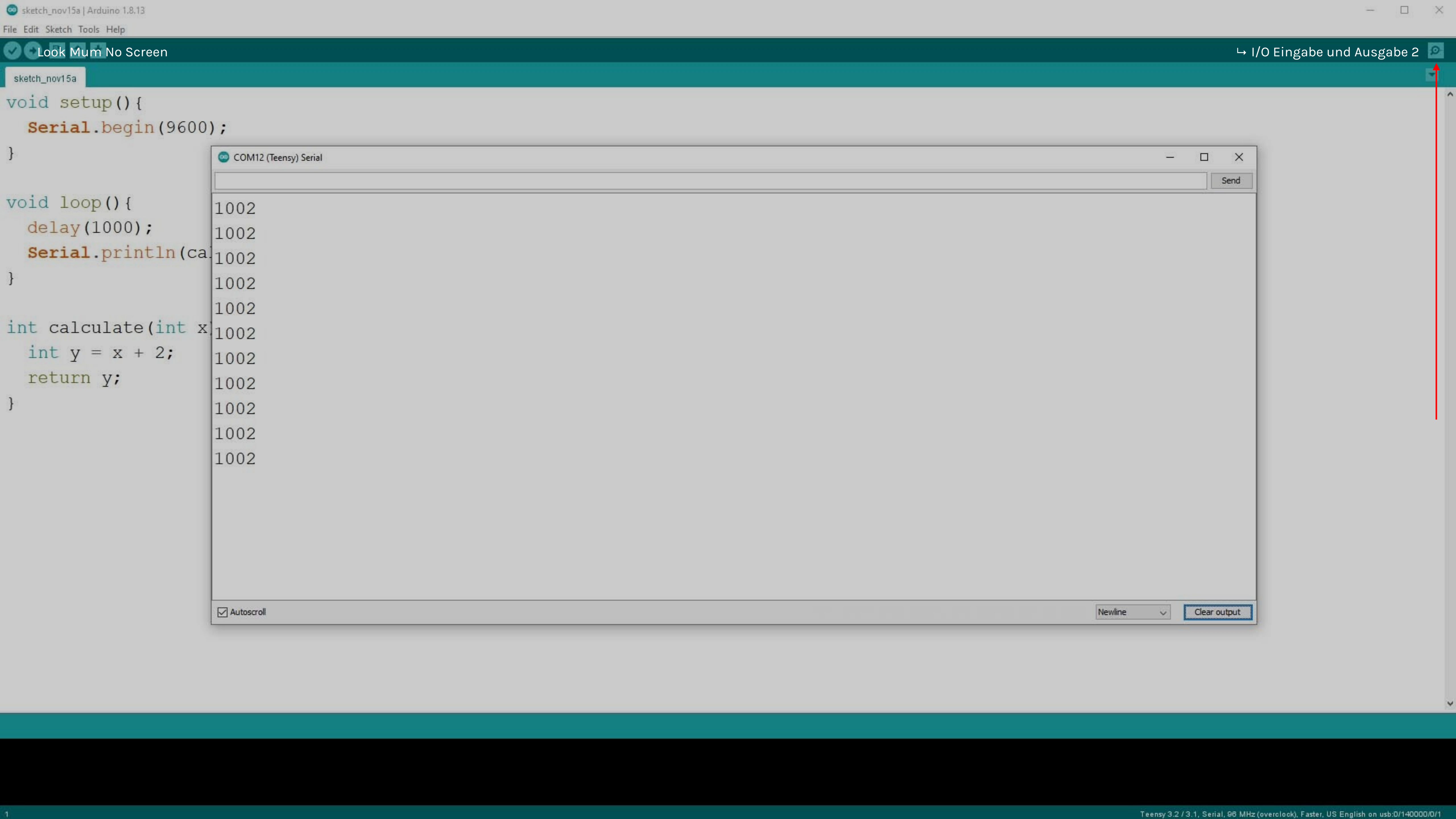
Dies geschieht, indem wir die Parameter auf die gleiche Weise wie
im Code definieren, jetzt aber in den Klammern nach dem
Funktionsnamen.

```
int calculate(int x){  
    int y = x + 2;  
    return y;  
}
```

```
calculate(1000);
```

Unsere Funktion wird aufgerufen, führt die Operation durch und
gibt uns den Wert **102** zurück.





```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  delay(1000);  
  Serial.println(1002);  
}  
  
int calculate(int x) {  
  int y = x + 2;  
  return y;  
}
```


COM12 (Teensy) Serial

1002
1002
1002
1002
1002
1002
1002
1002
1002
1002
1002

☒ Autoscroll Newline Clear output

Arithmetische
Operatoren:

Diese umfassen die **Addition, Subtraktion, Multiplikation und Division.**

```
int calculate(int x){  
    int y = x + 2;   
    return y;  
}
```

```
float value = x + 3.4;    // Addition  
int number = y - 5;      // Subtraktion  
float area = z * 23.4 ;  // Multiplikation  
int size = s / 5;        // Division
```

Arithmetische
Operatoren:

Ein weiterer wichtiger Operator ist: **Modulo (%)**. Er berechnet den Rest, wenn eine Zahl durch eine andere geteilt wird. Dieser wird meist dazu verwendet, um eine Variable in einem bestimmten Wertebereich zu halten.

```
int x = 0;

x = 7 % 5;    // x ist jetzt 2
x = 9 % 5;    // x ist jetzt 4
x = 5 % 5;    // x ist jetzt 0
x = 4 % 5;    // x ist jetzt 4
```

Achtung: Der Modulo-Operator arbeitet nicht float-Werten.

Weitere
Datentypen:

Ein Array beschreibt eine Sammlung von Werten auf die mit einer Indexnummer zugegriffen wird. Jeder Wert in dem Array kann aufgerufen werden, indem man den Namen des Arrays und die Indexnummer des Wertes abfragt. **Die Indexnummer fängt bei einem Array immer bei 0 an.**

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; ←
```

Weitere
Datentypen:

Ein Array beschreibt eine Sammlung von Werten auf die mit einer Indexnummer zugegriffen wird. Jeder Wert in dem Array kann aufgerufen werden, indem man den Namen des Arrays und die Indexnummer des Wertes abfragt. **Die Indexnummer fängt bei einem Array immer bei 0 an.**

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
                        ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
                    Indexnummern: 0 1 2 3 4 5 6 7 8 9
```

Weitere
Datentypen:

Ein Array beschreibt eine Sammlung von Werten auf die mit einer Indexnummer zugegriffen wird. Jeder Wert in dem Array kann aufgerufen werden, indem man den Namen des Arrays und die Indexnummer des Wertes abfragt. **Die Indexnummer fängt bei einem Array immer bei 0 an.**

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

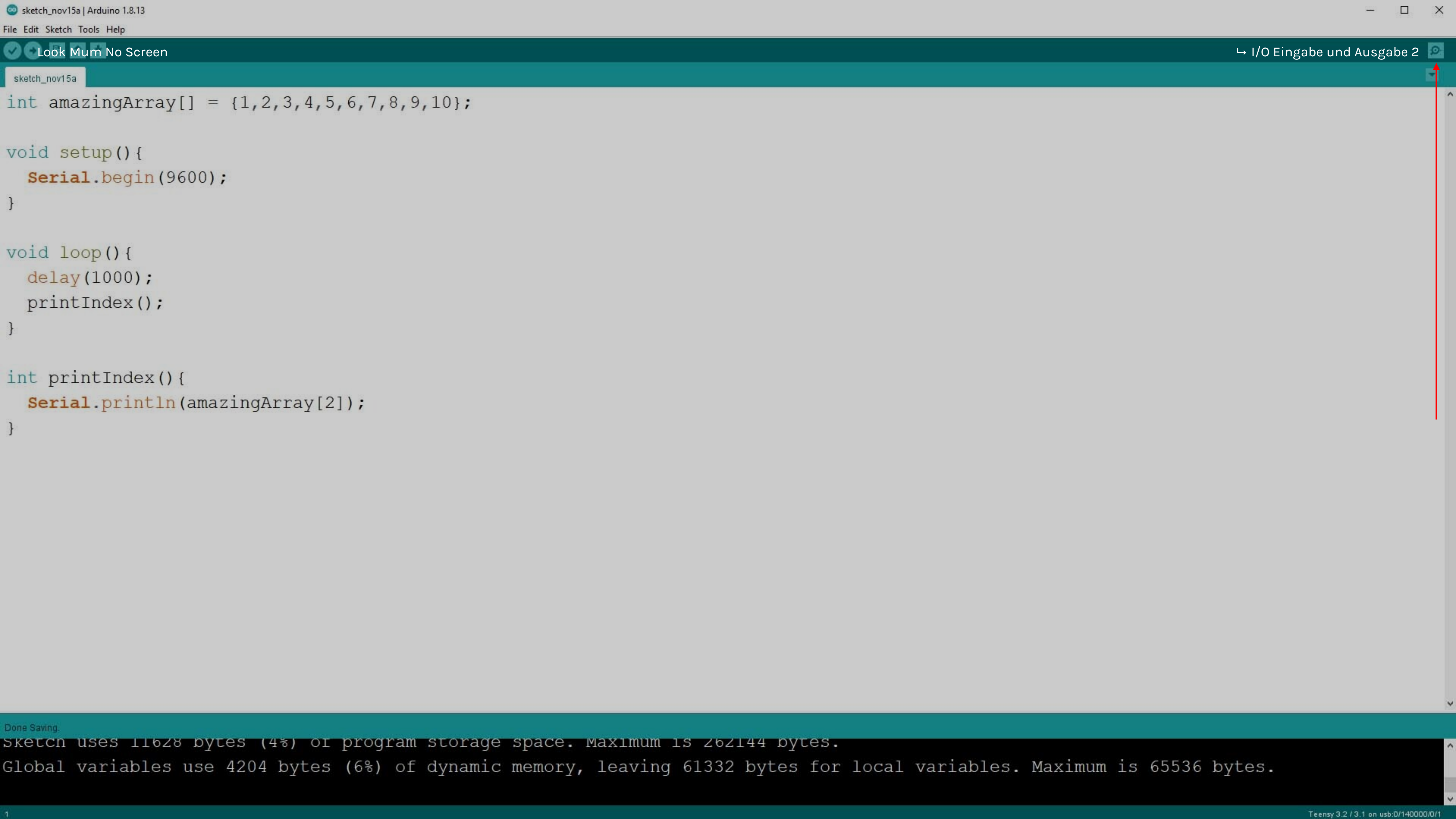
```
void setup(){  
  Serial.begin(9600);  
  // put your setup code here, to run once:  
}
```

```
void loop(){  
  delay(1000);  
  printIndex();  
  // put your main code here, to run repeatedly:  
}
```

```
void printIndex(){  
  Serial.println(amazingArray[2]);  
}
```



The diagram consists of three red arrows indicating the flow of execution. The first arrow starts at the `printIndex();` line in the `loop()` function and points to the `printIndex()` function definition. The second arrow starts at the `Serial.println(amazingArray[2]);` line inside the `printIndex()` function and points back to the `printIndex();` line in the `loop()` function. The third arrow starts at the `Serial.println(amazingArray[2]);` line and points to the right, indicating output to the serial monitor.



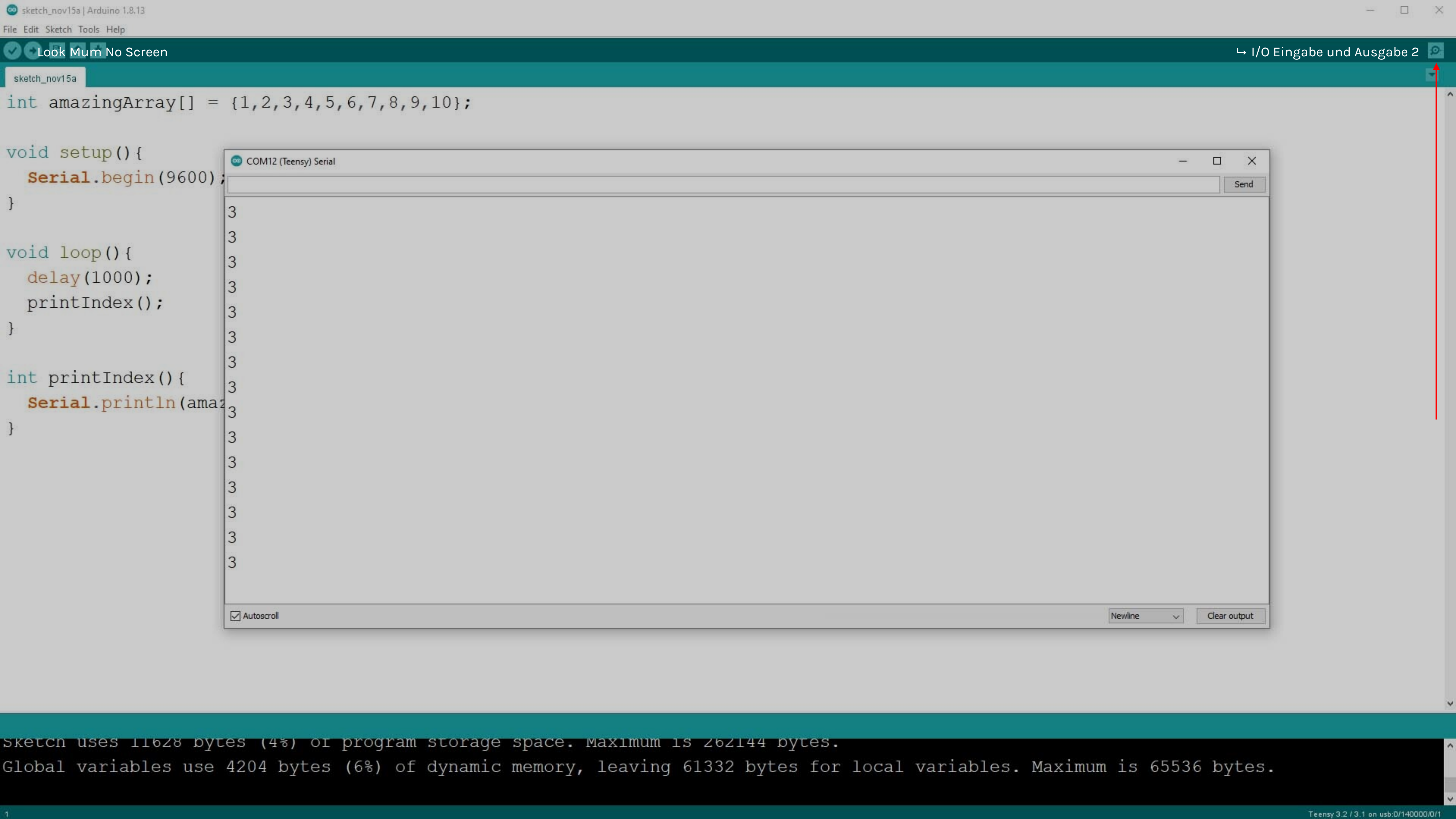
```
sketch_nov15a | Arduino 1.8.13
File Edit Sketch Tools Help
Look Mum No Screen
I/O Eingabe und Ausgabe 2
sketch_nov15a
int amazingArray[] = {1,2,3,4,5,6,7,8,9,10};

void setup(){
  Serial.begin(9600);
}

void loop(){
  delay(1000);
  printIndex();
}

int printIndex(){
  Serial.println(amazingArray[2]);
}

Done Saving.
Sketch uses 11628 bytes (4%) of program storage space. Maximum is 262144 bytes.
Global variables use 4204 bytes (6%) of dynamic memory, leaving 61332 bytes for local variables. Maximum is 65536 bytes.
Teensy 3.2 / 3.1 on usb:0/140000/0/1
```



Weitere
Datentypen:

Ein Array beschreibt eine Sammlung von Werten auf die mit einer Indexnummer zugegriffen wird. Jeder Wert in dem Array kann aufgerufen werden, indem man den Namen des Arrays und die Indexnummer des Wertes abfragt. **Die Indexnummer fängt bei einem Array immer bei 0 an.**

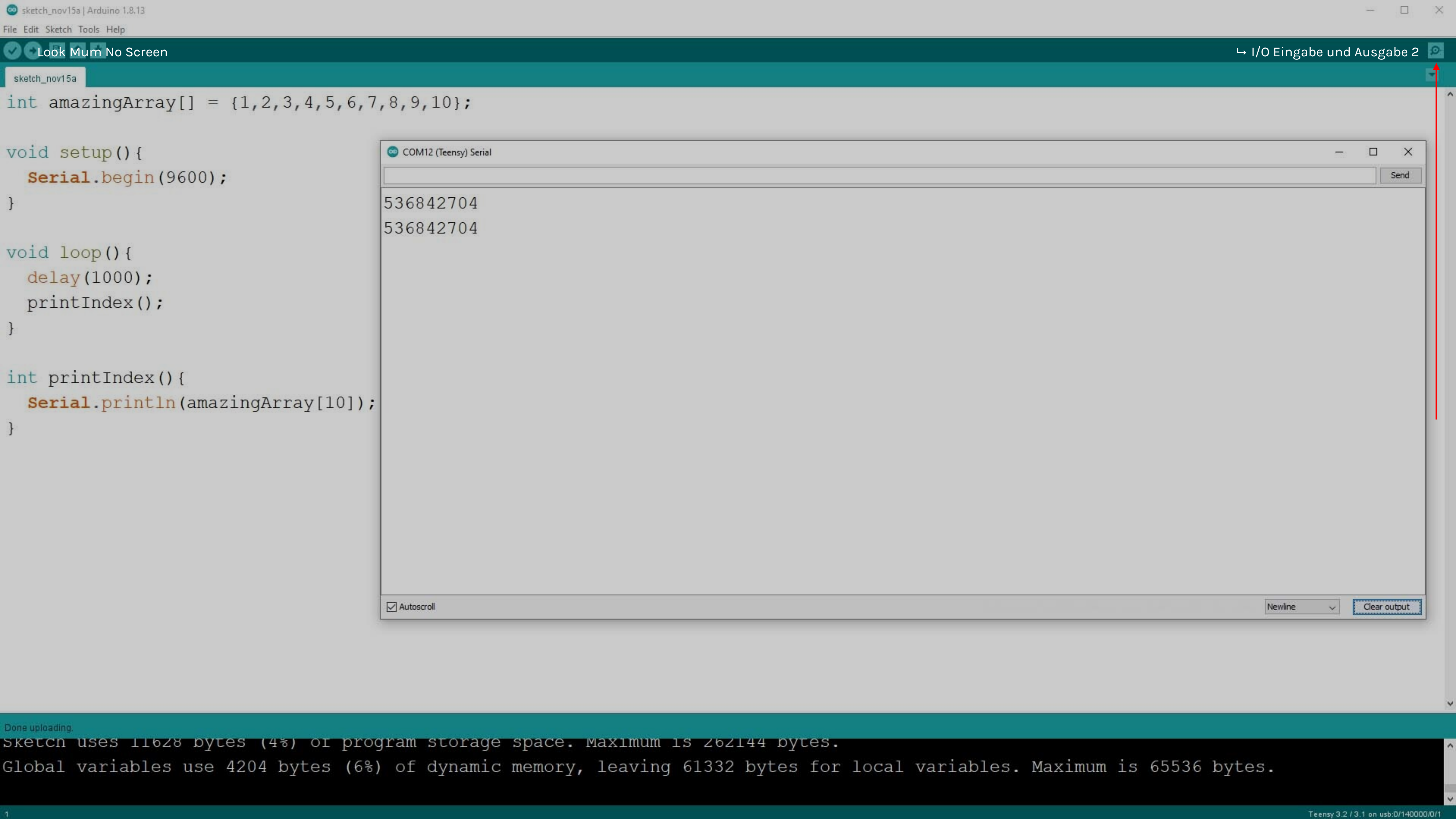
```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
void setup(){  
  Serial.begin(9600);  
  // put your setup code here, to run once:  
}
```

```
void loop(){  
  delay(1000);  
  printIndex();  
  // put your main code here, to run repeatedly:  
}
```

```
void printIndex(){  
  Serial.println(amazingArray[10]);  
}
```





```
sketch_nov15a

int amazingArray[] = {1,2,3,4,5,6,7,8,9,10};

void setup() {
  Serial.begin(9600);
}

void loop() {
  delay(1000);
  printIndex();
}

int printIndex() {
  Serial.println(amazingArray[10]);
}
```

COM12 (Teensy) Serial

536842704
536842704

☒ Autoscrol Newline Clear output

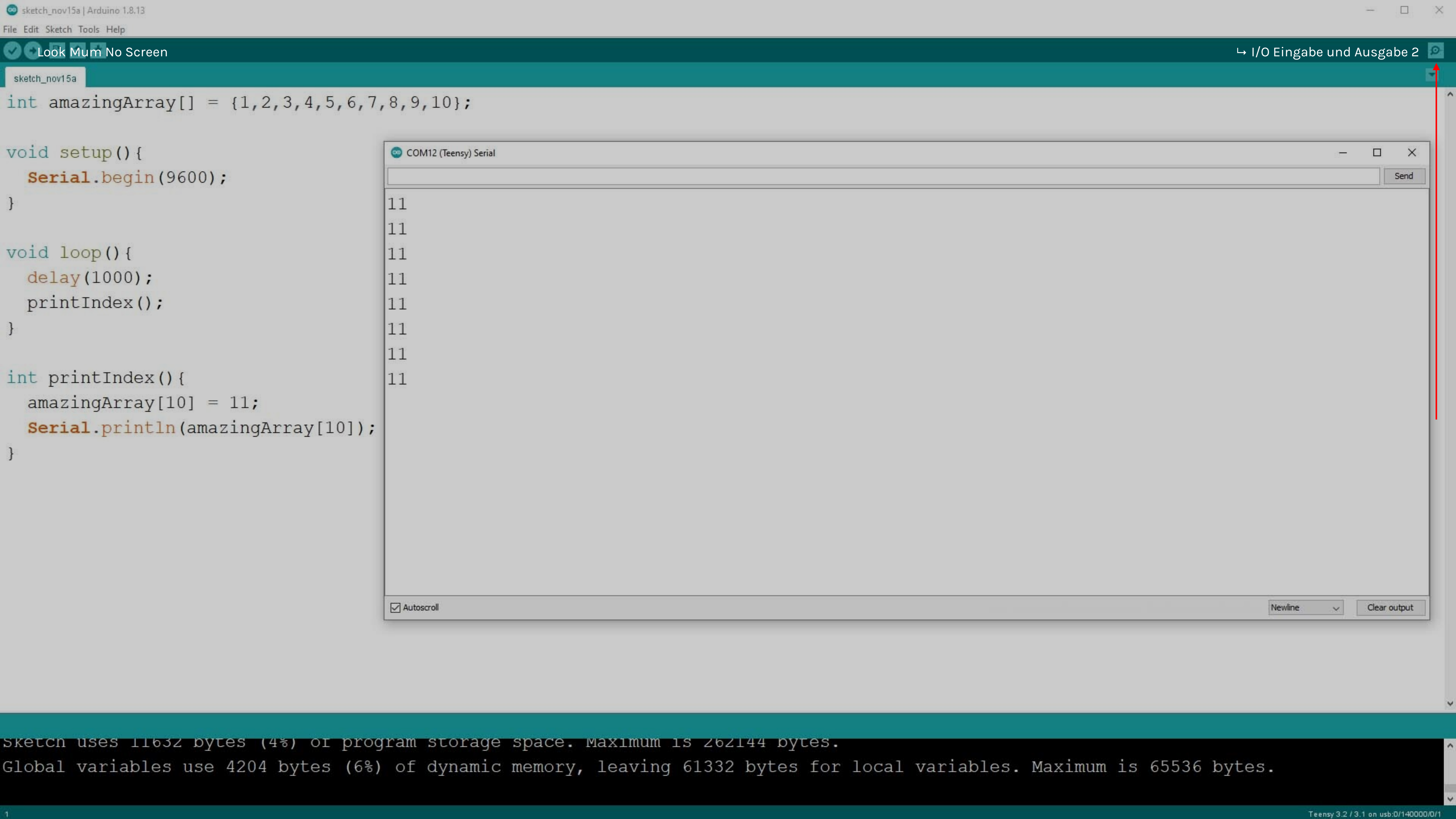
Weitere
Datentypen:

Ein Array beschreibt eine Sammlung von Werten auf die mit einer Indexnummer zugegriffen wird. Jeder Wert in dem Array kann aufgerufen werden, indem man den Namen des Arrays und die Indexnummer des Wertes abfragt. **Die Indexnummer fängt bei einem Array immer bei 0 an.**

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
void loop(){  
  delay(1000);  
  printIndex();  
  // put your main code here, to run repeatedly:  
}
```

```
void printIndex(){  
  amazingArray[10] = 11;           // Wert 11 an Stelle "10" zuweisen  
  Serial.println(amazingArray[10]); ←  
}
```



```
int amazingArray[] = {1,2,3,4,5,6,7,8,9,10};
```

```
void setup() {  
  Serial.begin(9600);  
}
```

```
void loop() {  
  delay(1000);  
  printIndex();  
}
```

```
int printIndex() {  
  amazingArray[10] = 11;  
  Serial.println(amazingArray[10]);  
}
```

COM12 (Teensy) Serial

11
11
11
11
11
11
11

☒ Autoscroll Newline Clear output

Sketch uses 11632 bytes (4%) of program storage space. Maximum is 262144 bytes.

Global variables use 4204 bytes (6%) of dynamic memory, leaving 61332 bytes for local variables. Maximum is 65536 bytes.

Weitere
Datentypen:

Ein Array beschreibt eine Sammlung von Werten auf die mit einer Indexnummer zugegriffen wird. Jeder Wert in dem Array kann aufgerufen werden, indem man den Namen des Arrays und die Indexnummer des Wertes abfragt. **Die Indexnummer fängt bei einem Array immer bei 0 an.**

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
void loop(){  
  delay(1000);  
  printIndex();  
  // put your main code here, to run repeatedly:  
}
```

```
void printIndex(){  
  amazingArray[10] = 11;           // Wert 11 an Stelle "10" zuweisen  
  int value = amazingArray[10];    // 11 in Variable value speichern  
  Serial.println(amazingArray[10]);  
}
```

Die for-Schleife:


Eine for-Schleife wird dazu verwendet, um einen Block von Anweisungen eine festgelegte Anzahl von Wiederholungen durchlaufen zu lassen. Dabei wird oft ein **Iterator** (ein Zähler) verwendet, um die Schleife ansteigen und später beenden zu lassen.

```
for (Initialisierung; Bedingung; Inkrement){  
    // schreibe deine Anweisungen hier:  
  
}
```

Die for-Schleife:

Im sogenannten Header einer for-Schleife stehen drei Elemente, jeweils getrennt durch ein Semikolon.

```
for (Initialisierung; Bedingung; Inkrement){  
    // schreibe deine Anweisungen hier:  
  
}
```



Initialisierung: Hier wird einmalig eine **lokale Variable** erstellt.


Bedingung: Bei jedem Durchlauf der for-Schleife wird die Bedingung geprüft. Solange die Bedingung wahr ist, läuft die Schleife weiter, und alle Anweisungen in der Schleife werden ausgeführt. Ist die Bedingung nicht mehr wahr, wird die Schleife verlassen.

Inkrement: Wird hochgezählt, wenn die **Bedingung** wahr ist.

Die for-Schleife:

Im Header **initialisieren** wir einmalig **int i** und wir definieren eine **Bedingung**.

```
for (int i = 0; i < 10;){  
    i++;  
    Serial.println(i);  
}
```

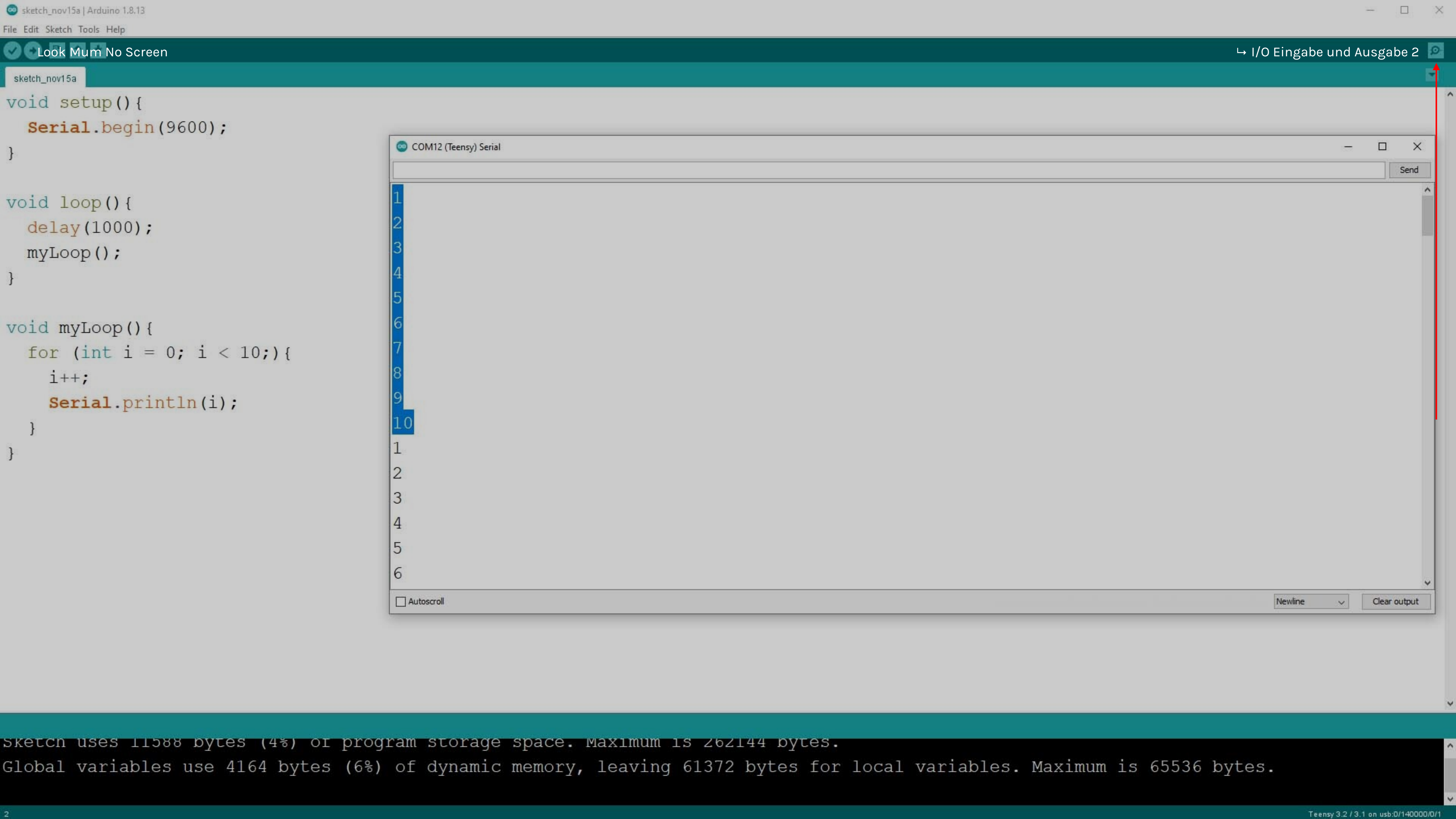


Die for-Schleife:

Im Header **initialisieren** wir einmalig **int i** und wir definieren eine **Bedingung**.

```
for (int i = 0; i < 10;){  
    i++;          // Erhöht den Wert einer Variablen um 1.  
    Serial.println(i);  
}
```

1
2
3
4
5
6
7
8
9
10
1
2
3



```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  delay(1000);  
  myLoop();  
}  
  
void myLoop() {  
  for (int i = 0; i < 10;) {  
    i++;  
    Serial.println(i);  
  }  
}
```

COM12 (Teensy) Serial

Send

1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6

☐ Autoscroll

Newline ▾

Clear output

Sketch uses 11588 bytes (4%) of program storage space. Maximum is 262144 bytes.
Global variables use 4164 bytes (6%) of dynamic memory, leaving 61372 bytes for local variables. Maximum is 65536 bytes.

Die for-Schleife:
Achtung!

Im Header **initialisieren** wir einmalig **int i** und wir definieren eine **Bedingung**.

```
for (int i = 0; i < 10;){  
  Serial.println(i);  
  i++;      // Erhöht den Wert einer Variablen um 1.  
}
```

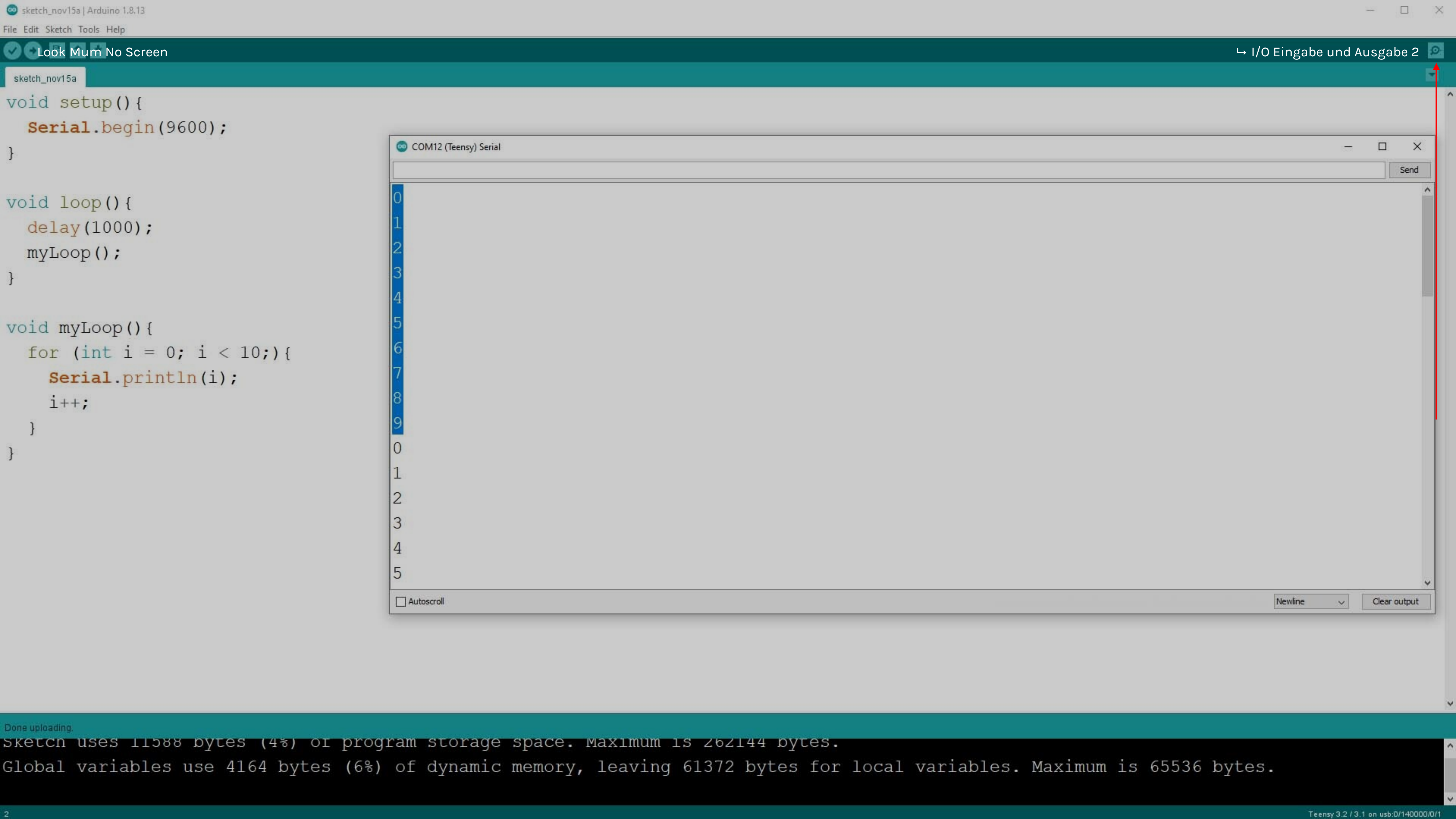
0 ←
1
2
3
4
5
6
7
8
9 ←
0
1
2

Die for-Schleife:

Immer darauf achten, die Variable erst zum Schluss zu verändern.
Unter Umständen bricht sonst die Logik in der Schleife.

```
for (int i = 0; i < 10;){  
    Serial.println(i);  
    i++;  
}
```



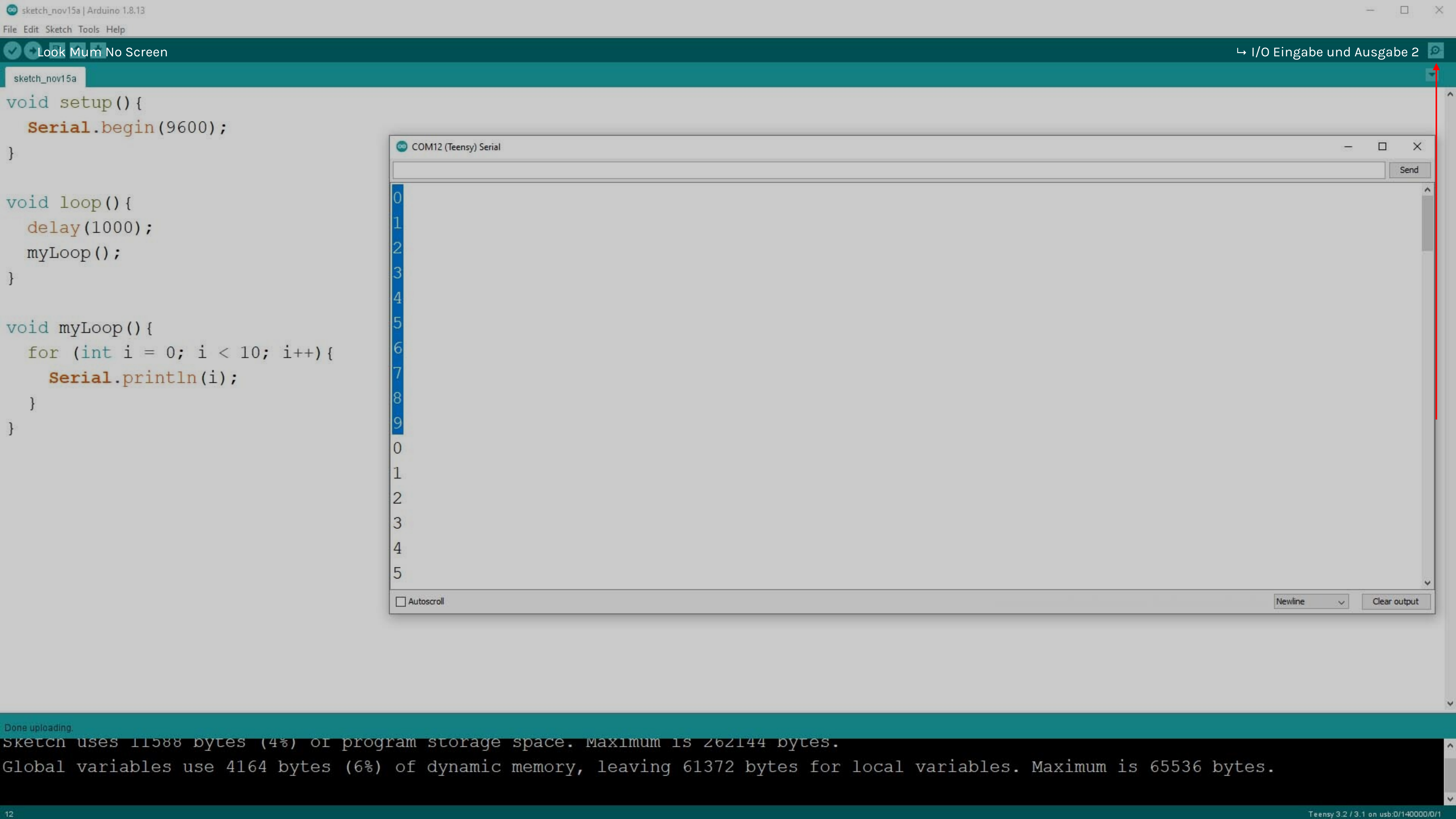


Die for-Schleife:

Im Header **initialisieren** wir einmalig **int i** und wir definieren eine **Bedingung**. Weiterhin fügen wir hier ein Inkrement hinzu.

```
for (int i = 0; i < 10; i++){  
    Serial.println(i);  
}
```

0
1
2
3
4
5
6
7
8
9
0
1
2



```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  delay(1000);  
  myLoop();  
}  
  
void myLoop() {  
  for (int i = 0; i < 10; i++) {  
    Serial.println(i);  
  }  
}
```

Done uploading.
Sketch uses 11588 bytes (4%) of program storage space. Maximum is 262144 bytes.
Global variables use 4164 bytes (6%) of dynamic memory, leaving 61372 bytes for local variables. Maximum is 65536 bytes.

Die for-Schleife:

Im Header **initialisieren** wir einmalig **int i** und wir definieren eine **Bedingung**. Weiterhin fügen wir hier ein Inkrement hinzu. Diese for-Schleife wird insgesamt 5 Mal ausgeführt.

```
#define ledPin 5
```

```
for (int i = 0; i < 5; i++){  
    digitalWrite(ledPin, HIGH); // ledPin wird eingeschaltet.  
    delay(500);                // Pause für eine halbe Sekunde.  
    digitalWrite(ledPin, LOW);  // ledPin wird ausgeschaltet.  
    delay(500);                // Pause für eine halbe Sekunde.  
}
```


Die for-Schleife:

Im Header **initialisieren** wir einmalig **int i** und wir definieren eine **Bedingung**. Weiterhin fügen wir hier ein Inkrement hinzu. Diese for-Schleife wird insgesamt 5 Mal ausgeführt.

```
#define ledPin 5

for (int i = 0; i < 5; i++){
    digitalWrite(ledPin, HIGH); // ledPin wird eingeschaltet.
    delay(500);                 // Pause für eine halbe Sekunde.
    digitalWrite(ledPin, LOW);  // ledPin wird ausgeschaltet.
    delay(500);                 // Pause für eine halbe Sekunde.
    Serial.println(i);
}

0
1
2
3
4
0
1
2
```

Arrays und for-Schleifen:

Arrays lassen sich hervorragend innerhalb von for-Schleifen bearbeiten. Achtung: Dabei wird die **Indexnummer** eines Arrays als **Iterator** (Zähler) verwendet.

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```


```
for (int i = 0; i < 5; i++){  
    Serial.println(amazingArray[i]);  
}
```

Arrays und for-Schleifen:

Arrays lassen sich hervorragend innerhalb von for-Schleifen bearbeiten. Achtung: Dabei wird die **Indexnummer** eines Arrays als **Iterator** (Zähler) verwendet.

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
for (int i = 0; i < 5; i++){  
    Serial.println(amazingArray[i]);  
}
```




Arrays und for-Schleifen:

Arrays lassen sich hervorragend innerhalb von for-Schleifen bearbeiten. Achtung: Dabei wird die **Indexnummer** eines Arrays als **Iterator** (Zähler) verwendet.

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
for (int i = 0; i < 5; i++){  
    Serial.println(amazingArray[i]);  
}
```



Arrays und for-Schleifen:

Arrays lassen sich hervorragend innerhalb von for-Schleifen bearbeiten. Achtung: Dabei wird die **Indexnummer** eines Arrays als **Iterator** (Zähler) verwendet.

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```



```
for (int i = 0; i < 5; i++){ ←  
    Serial.println(amazingArray[i]);  
}
```

Arrays und for-Schleifen:

Arrays lassen sich hervorragend innerhalb von for-Schleifen bearbeiten. Achtung: Dabei wird die **Indexnummer** eines Arrays als **Iterator** (Zähler) verwendet.

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```



```
for (int i = 0; i < 5; i++){ ←  
    Serial.println(amazingArray[i]);  
}
```

Arrays und for-Schleifen:

Arrays lassen sich hervorragend innerhalb von for-Schleifen bearbeiten. Achtung: Dabei wird die **Indexnummer** eines Arrays als **Iterator** (Zähler) verwendet.

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
                        ↑  ↑  ↑  ↑  
for (int i = 0; i < 5; i++){ ←  
    Serial.println(amazingArray[i]);  
}
```

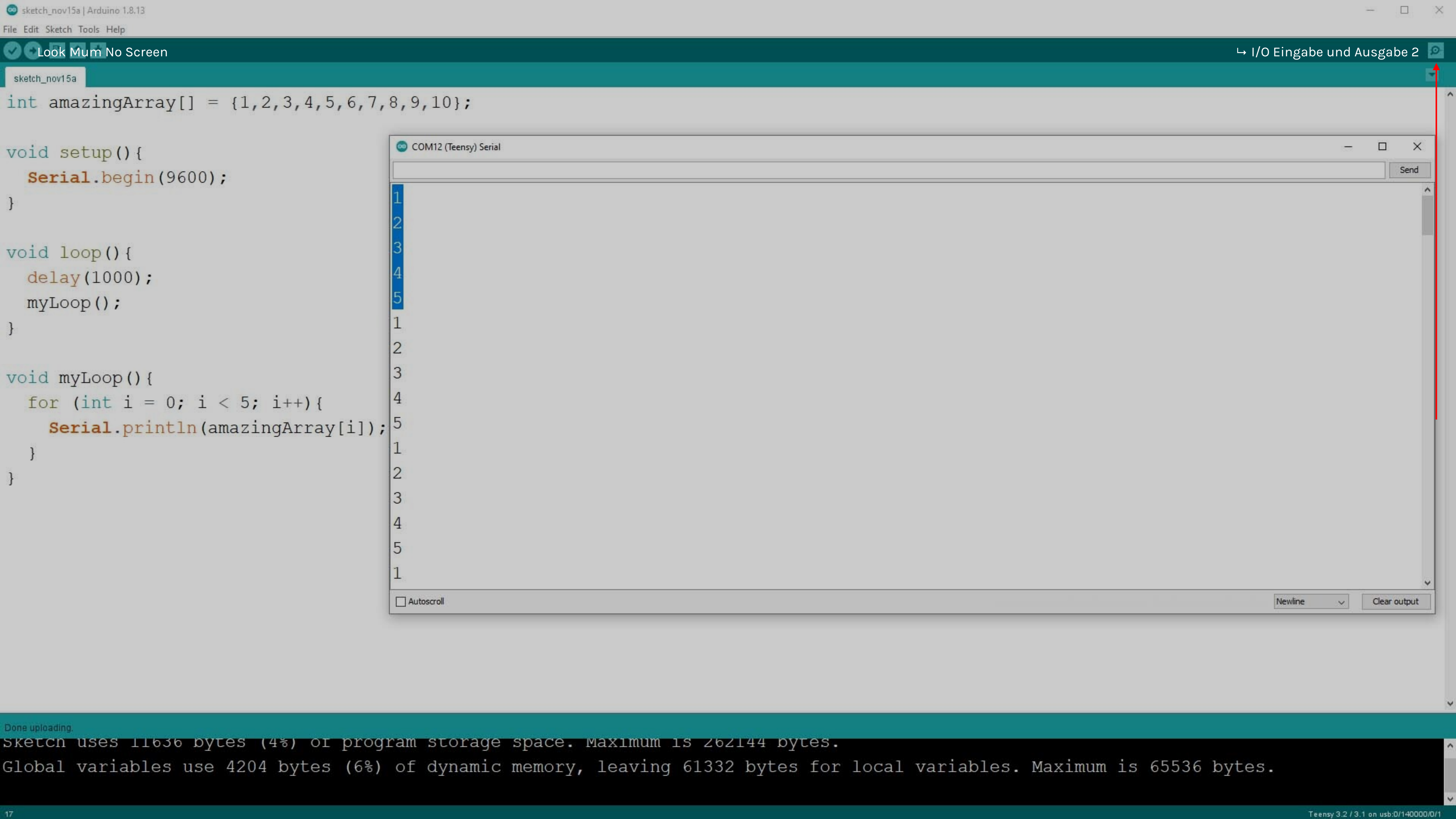
Arrays und for-Schleifen:

Arrays lassen sich hervorragend innerhalb von for-Schleifen bearbeiten. Achtung: Dabei wird die **Indexnummer** eines Arrays als **Iterator** (Zähler) verwendet.

```
int amazingArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
                        ↑ ↑ ↑ ↑ ↑
```

```
for (int i = 0; i < 5; i++){ ←
    Serial.println(amazingArray[i]);
}
```

1
2
3
4
5
1
2



Arrays und for-Schleifen:

Wir initialisieren in diesem Beispiel nun insgesamt 5 pins mit der Hilfe eines Arrays und definieren später in der setup()-Funktion ihren Pin Mode.

```
const int pinArray [ ] = {9, 10, 11, 12, 13};
```

Arrays und for-Schleifen:

Wir initialisieren in diesem Beispiel nun insgesamt 5 pins mit der Hilfe eines Arrays und definieren später in der setup()-Funktion ihren Pin Mode.

—————→ `const int pinArray [] = {9, 10, 11, 12, 13};`

Kleiner Exkurs: Werte die sich während der Laufzeit nicht ändern, wie die Belegung der Pins, können auch mit einem **const** initialisiert werden.

Arrays und for-Schleifen:

Wir initialisieren in diesem Beispiel nun insgesamt 5 pins mit der Hilfe eines Arrays und definieren später in der setup()-Funktion ihren Pin Mode.

```
const int pinArray [ ] = {9, 10, 11, 12, 13};
```

```
void setup(){  
  Serial.begin(9600);
```

```
  for(int i = 0; i < 5; i++){  
    pinMode(pinArray[i], INPUT);  
    Serial.println(pinArray[i]);  
  }
```

```
  // put your setup code here, to run once:  
}
```

```
9  
10  
11  
12  
13
```




while-Schleife:

Im Gegensatz zu einer for-Schleife, werden while-Schleifen unbegrenzt wiederholt bis eine Bedingung, innerhalb der Klammern, falsch ist oder ihren Zustand ändert. Wenn wir also eine Variable in der while-Schleife testen wollen, muss diese sich an einer anderen Stelle im Programm ändern, sonst endet die while-Schleife nicht.

```
int value = 0;
```

```
void setup(){  
  Serial.begin(9600);  
}
```

```
void loop(){  
  delay(500);  
  while (value < 5) {  
    Serial.println("Ja moin");  
  }  
}
```

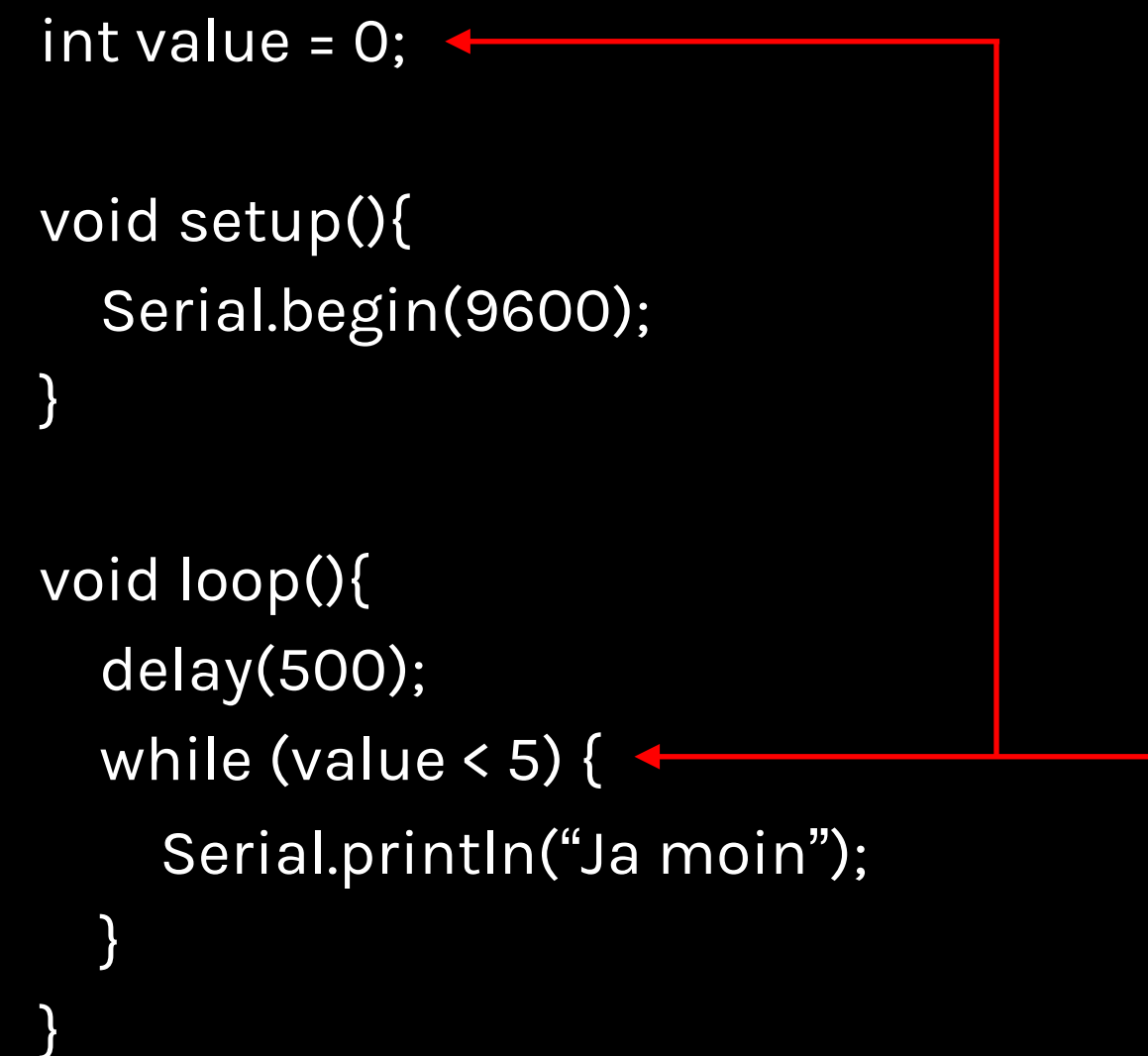
while-Schleife:

Im Gegensatz zu einer for-Schleife, werden while-Schleifen unbegrenzt wiederholt bis eine Bedingung, innerhalb der Klammern, falsch ist oder ihren Zustand ändert. Wenn wir also eine Variable in der while-Schleife testen wollen, muss diese sich an einer anderen Stelle im Programm ändern, sonst endet die while-Schleife nicht.

```
int value = 0;

void setup(){
  Serial.begin(9600);
}

void loop(){
  delay(500);
  while (value < 5) {
    Serial.println("Ja moin");
  }
}
```



while-Schleife:

Im Gegensatz zu einer for-Schleife, werden while-Schleifen unbegrenzt wiederholt bis eine Bedingung, innerhalb der Klammern, falsch ist oder ihren Zustand ändert. Wenn wir also eine Variable in der while-Schleife testen wollen, muss diese sich an einer anderen Stelle im Programm ändern, sonst endet die while-Schleife nicht.

```
int value = 0;

void setup(){
  Serial.begin(9600);
}

void loop(){
  delay(500);
  while (value < 5) {
    Serial.println("Ja moin");
  }
}
```

Ja moin
Ja moin
Ja moin
Ja moin
Ja moin
Ja moin
Ja moin
Ja moin

Ja moin
Ja moin
Ja moin


while-Schleife:

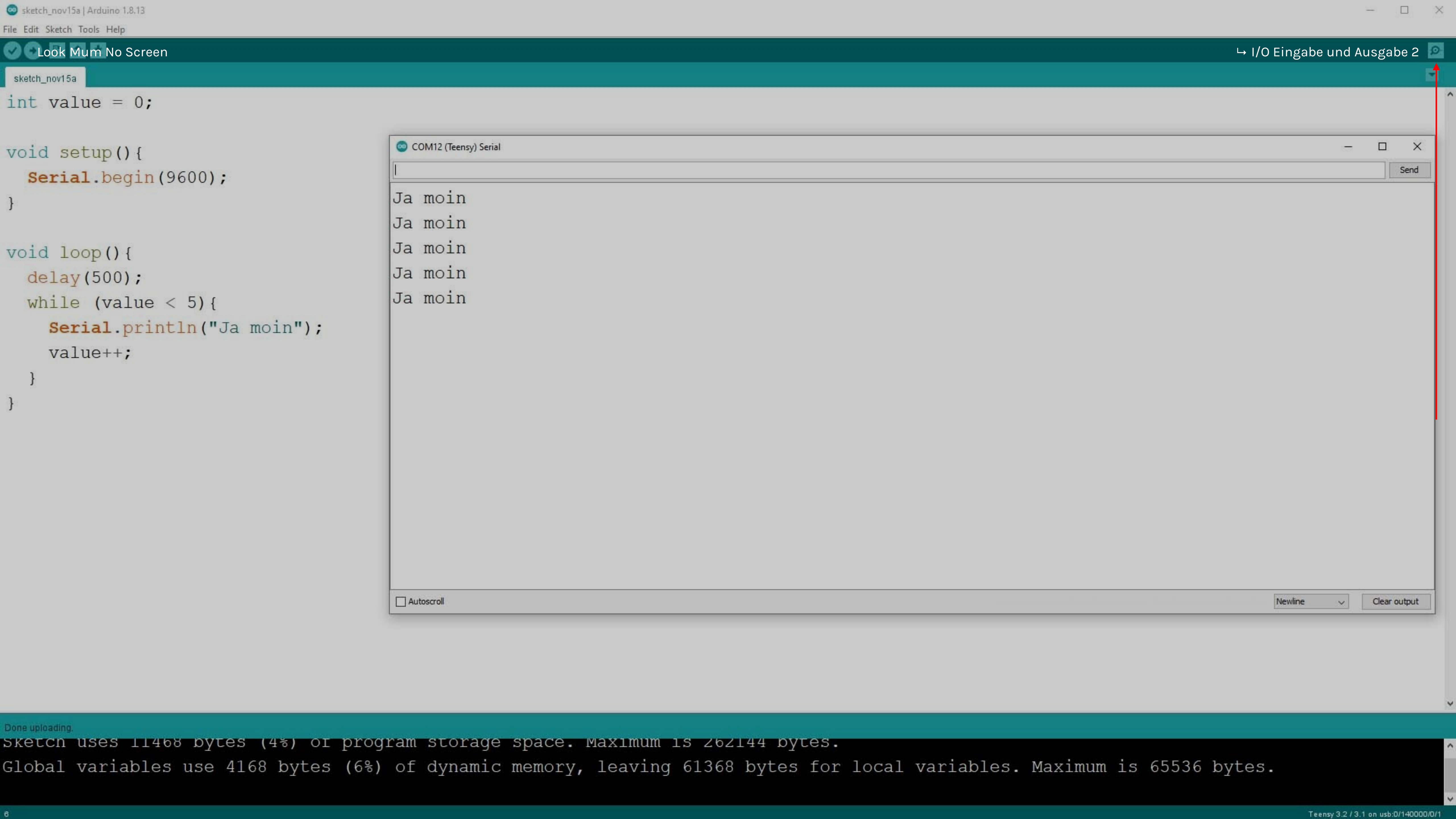
Im Gegensatz zu einer for-Schleife, werden while-Schleifen unbegrenzt wiederholt bis eine Bedingung, innerhalb der Klammern, falsch ist oder ihren Zustand ändert. Wenn wir also eine Variable in der while-Schleife testen wollen, muss diese sich an einer anderen Stelle im Programm ändern, sonst endet die while-Schleife nicht.

```
int value = 0;
```

```
void setup(){  
  Serial.begin(9600);  
}
```

```
void loop(){  
  delay(500);  
  while (value < 5) {  
    Serial.println("Ja moin");  
    value++;  
  }  
}
```





```
sketch_nov15a | Arduino 1.8.13
File Edit Sketch Tools Help
Look Mum No Screen
sketch_nov15a
int value = 0;

void setup() {
  Serial.begin(9600);
}


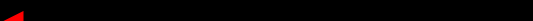
void loop() {
  delay(500);
  while (value < 5) {
    Serial.println("Ja moin");
    value++;
  }
}

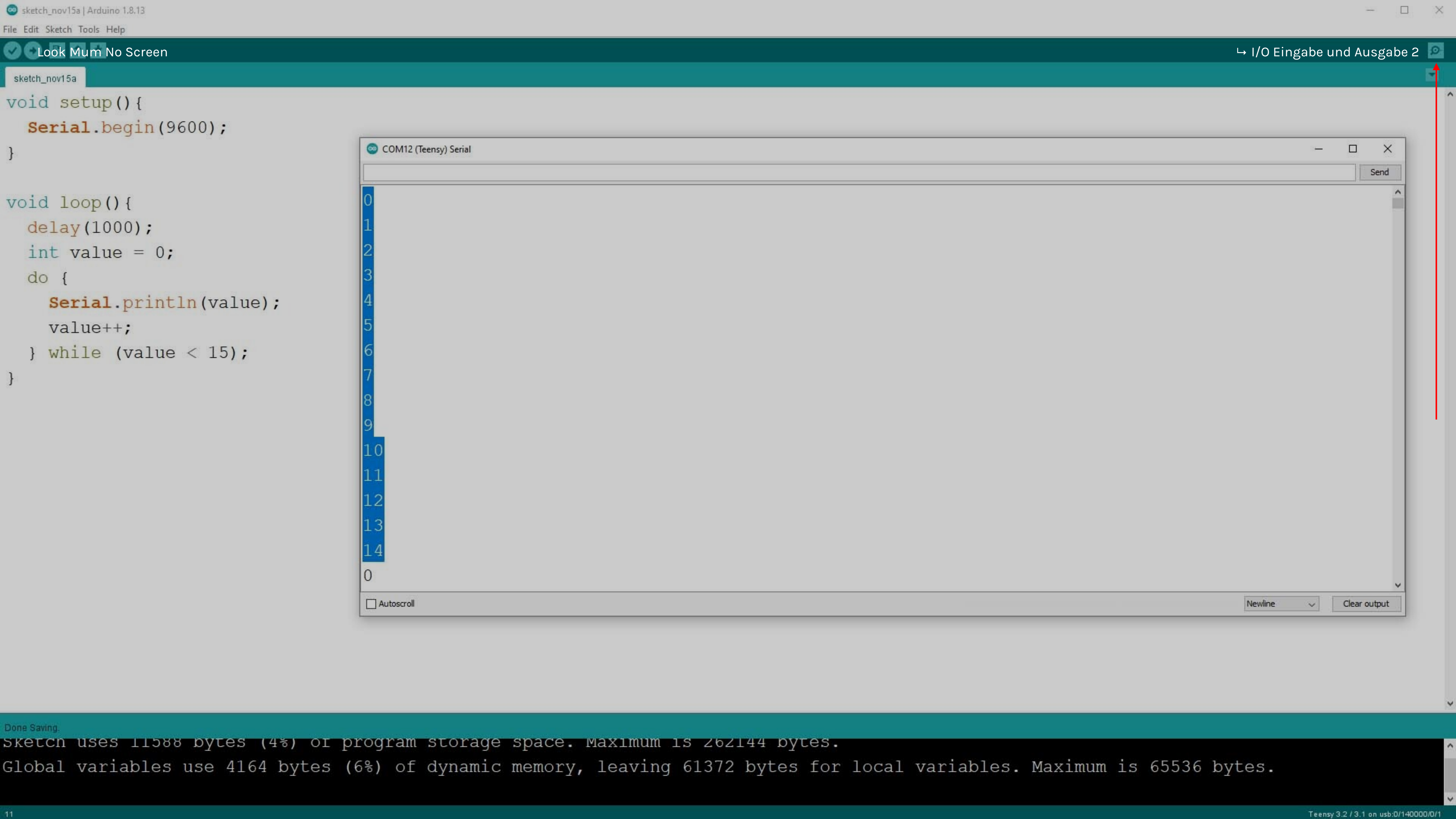
COM12 (Teensy) Serial
Ja moin
Ja moin
Ja moin
Ja moin
Ja moin

Autoscroll Newline Clear output
Done uploading.
Sketch uses 11468 bytes (4%) of program storage space. Maximum is 262144 bytes.
Global variables use 4168 bytes (6%) of dynamic memory, leaving 61368 bytes for local variables. Maximum is 65536 bytes.
Teensy 3.2 / 3.1 on usb:0/140000/0/1
```

do-while-Schleife:

Im direkten Vergleich zur while-Schleife ist die do-while-Schleife eine endgesteuerte Schleife. Der einzige Unterschied ist, dass das Prüfen der Bedingung am Ende der Schleife stattfindet. Das bedeutet, die Schleife läuft immer mindestens einmal.

```
void setup(){  
  Serial.begin(9600);  
  int value = 0;   
  
  do {  
    Serial.println(value);  
    value++;   
  } while (value < 15);  
}
```



```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  delay(1000);  
  int value = 0;  
  do {  
    Serial.println(value);  
    value++;  
  } while (value < 15);  
}
```

COM12 (Teensy) Serial

Send

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
0

☐ Autoscrol Newline Clear output

Weitere nützliche
Funktionen:

random(min, max);

Die Random-Funktion generiert Zufallszahlen.

```
int x = random(0, 10);           // z.B. x = 4
```

pow(base, exponent);

Sie berechnet den Wert einer Zahl, die auf eine Potenz erhöht wird.

```
int x = pow(7,2);                // x = 49
```

map(value, fromLow, fromHigh, toLow, toHigh);

Sie bildet eine Zahl von einem Bereich in einen anderen ab.

value: Die Nummer, die zugeordnet werden soll.

fromLow: Die untere Grenze des aktuellen Wertebereich.

fromHigh: Die obere Grenze des aktuellen Wertebereich.

toLow: Die untere Grenze des zu erreichenden Wertebereich.

toHigh: Die obere Grenze des zu erreichenden Wertebereich.

```
void loop(){
```

```
  int val = analogRead(0);
```

```
  val = map(val, 0, 1023, 0, 255);
```

```
  Serial.println(val);
```

```
}
```



Aufgabe:

Die letzte Veranstaltung vor Weihnachten... Zwischenpräsentation!

Für die Zwischenpräsentation möchten wir wissen, wie euer System funktionieren soll. Konkret solltet ihr beantworten können:

Aus welchen Bauteilen und Komponenten soll euer System bestehen?

Welche Daten und Informationen möchtet ihr aus dem Internet abfragen?

Wie soll euer System reagieren, wenn eine Veränderung der Daten stattfindet?

Bitte beschäftigt euch weiterhin mit dem Servo Motor und dem Ultraschal Sensor.

Links: <https://www.arduino.cc/reference/de/>
<https://www.arduino.cc/reference/de/language/variables/data-types/array/>
<https://funduino.de/anleitung>

Material: <https://github.com/tmjns/Look-Mum-No-Screen>