




Einführung in Processing

Für die Programmierung in Processing steht eine integrierte Entwicklungsumgebung (**IDE** – engl. integrated development environment) zur Verfügung. Eine IDE ermöglicht das Erstellen von Computerprogrammen in einer Programmieroberfläche.

1. Gib in Processing die Anweisungen so ein, wie sie in der Abbildung unten dargestellt sind.

Lege die Verzeichnisstruktur  << Processing > Programme > siebzehnhundvier an.


- Speichere das Programm unter der Bezeichnung **siebzehnplusvier**.
(vgl. .\262-materialien\rechnung\sketch_01-siebzehnplusvier)

Processing legt für jedes Programm ein eigenes Verzeichnis an:  **sketch_01_siebzehnplusvier**

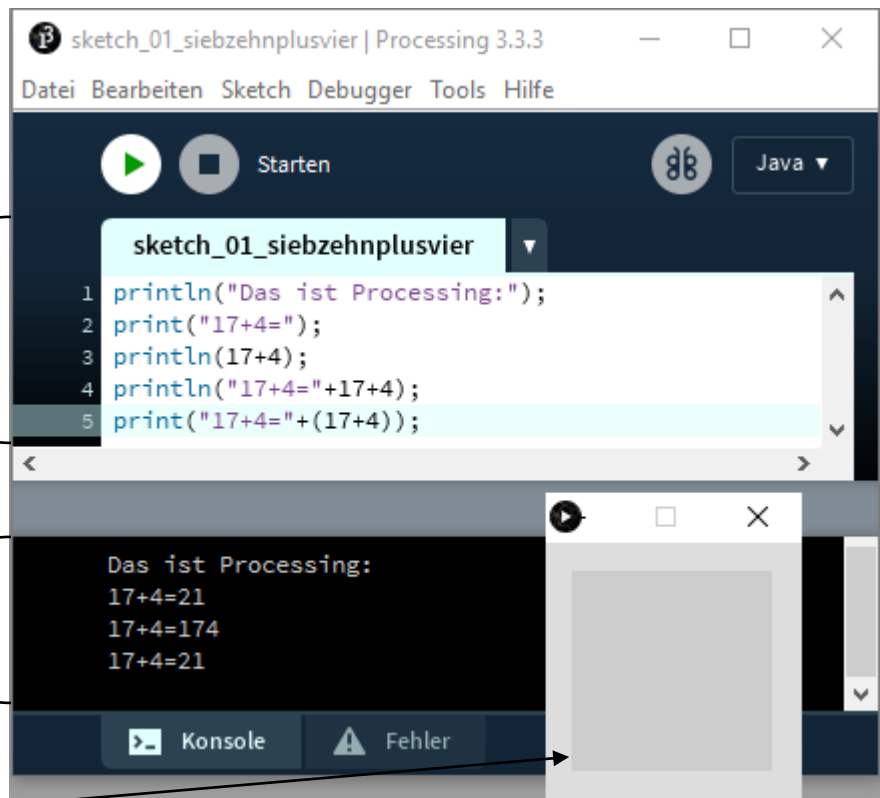
In diesem Verzeichnis ist die Programmdatei **sketch_01_siebzehnplusvier.pde**.
(pde steht für **processing development environment**).


Die Bezeichnung **sketch** (engl. für „Skizze“) soll darauf hindeuten, dass man in Processing schnell Zeichnungen erzeugen kann.

Das Programm wird in dem **Texteditor** eingegeben.

- Starte das Programm, indem du auf die Schaltfläche  klickst.

In der **Konsole** können Textausgaben angezeigt werden. Für Grafikausgaben wird nach dem Programmstart ein **Zeichenfenster** geöffnet.



- Beende das Programm, indem du auf die Schaltfläche  klickst.
- Vergleiche die Ausgaben in der Konsole mit den Anweisungen des Programms. Beschreibe die Wirkung der Anweisungen in den einzelnen Programmzeilen.

print: Ausgabe in der Konsole.

println: Erzeugt am Ende einen Zeilenumbruch (ln für „line“).

1: Gibt in der Konsole den Text in Anführungszeichen aus („Das ist Processing“).

2: Der Text „17+4=“ wird angezeigt.

3: Ohne Anführungszeichen wird das Ergebnis der Rechnung 17+4 ausgegeben.

4: Texte werden mit dem Zeichen + aneinandergehängt („verkettet“).

5: Zahlen werden mit dem Zeichen + addiert.



Variablen

2. Wie lautet die Anweisung für die Zeile 6? Ergänze und teste das Programm.
Speichere den Sketch als 02_siebzehnplusvier.
(vgl. .\262-materialien\rechnung\sketch_02_siebzehnplusvier_variable)

```
sketch_02_rechnung_variable
1 String Titel;
2 int Summe;
3 Titel="Processing mit Variablen:";
4 Summe=17+4;
5 println(Titel);
6 print("17+4="+Summe);
```

Processing mit Variablen:
17+4=21

Lösungsbeispiel für Nr. 3:

```
sketch_03_rechnung
1 int Produkt;
2 float Quotient;
3 Produkt=17*4;
4 Quotient=Produkt/5;
5 println("17*4="+Produkt);
6 print(Produkt+"/"+5="+Quotient);
```

17*4=68
68/5=13.0

3. Implementiere ein Programm, das zwei Zahlen multipliziert und dann das Produkt durch eine andere Zahl dividiert. Die Terme sollen in der Konsole angezeigt werden. Achte darauf, dass der Quotient keine ganze Zahl ist. Hinweis: Das Zeichen für Multiplikation ist *, für Division /.

- Ergänze den Algorithmus in dem Aktivitätsdiagramm.
- Codiere den Algorithmus in Processing und speichere den Sketch als 03_rechnung.
(vgl. .\262-materialien\rechnung\sketch_03_rechnung)

- Sieh dir das Ergebnis genau an. Was stellst du fest?

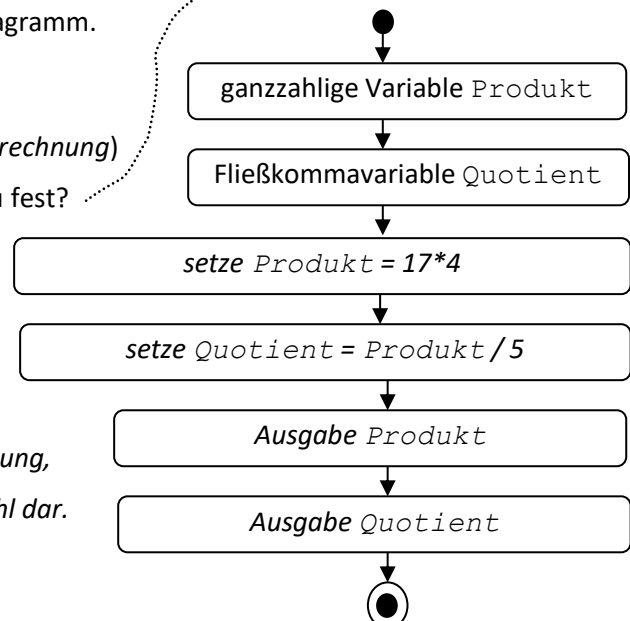
Die Nachkommastelle wird abgeschnitten.

- Kannst du dir einen Grund dafür vorstellen?

Das Produkt ist vom Datentyp integer, also eine ganze Zahl.

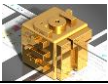
Processing stellt auch das Ergebnis der Rechnung, in der diese Variable vorkommt, als ganze Zahl dar.

Aktivitätsdiagramm:



Das Problem kann man mit einer **Funktion** beheben werden. Das ist eine Art von Unterprogramm, das man in dem eigentlichen Programm aufrufen kann. Eine Programmiersprache verfügt über viele Funktionen. Ein Beispiel ist die Funktion `float()`, die ganzzahlige Werte in Kommazahlen umwandelt. Statt `Quotient=Produkt/5;` kann man kann z. B. schreiben `Quotient=float(Produkt)/5;`. Dadurch wird der Wert der Variablen `Produkt` vor der Division in eine Kommazahl umgewandelt.

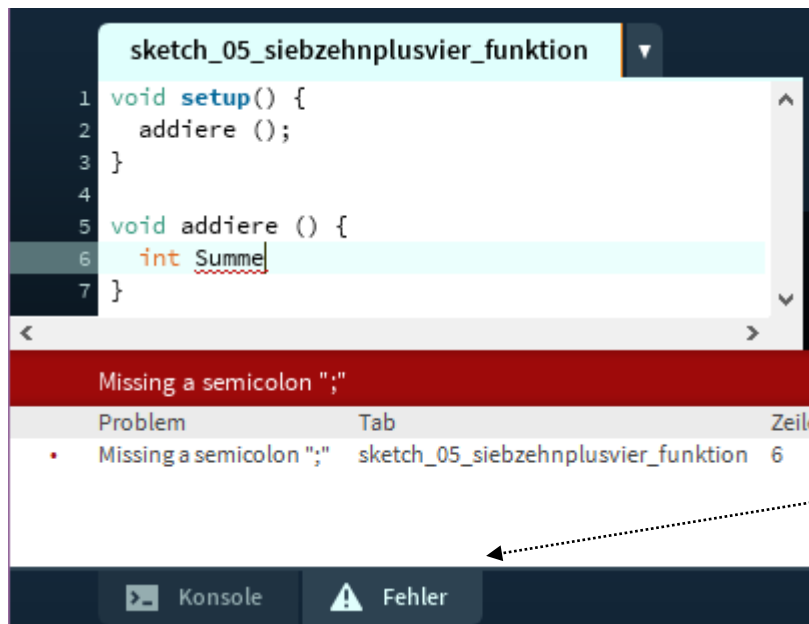
4. Ergänze dein Programm (04_rechnung_funktion).
(vgl. .\262-materialien\rechnung\sketch_04_rechnung_funktion)



Funktionen

5. Gib in Processing das Programm so ein, wie es in den Lerninhalten S. 3 beschrieben ist. Speichere das Programm unter der Bezeichnung `05_siebzehnplusvier_funktion`. (vgl. `.\262-materialien\rechnung\sketch_05_siebzehnplusvier_funktion`)

Hinweise:

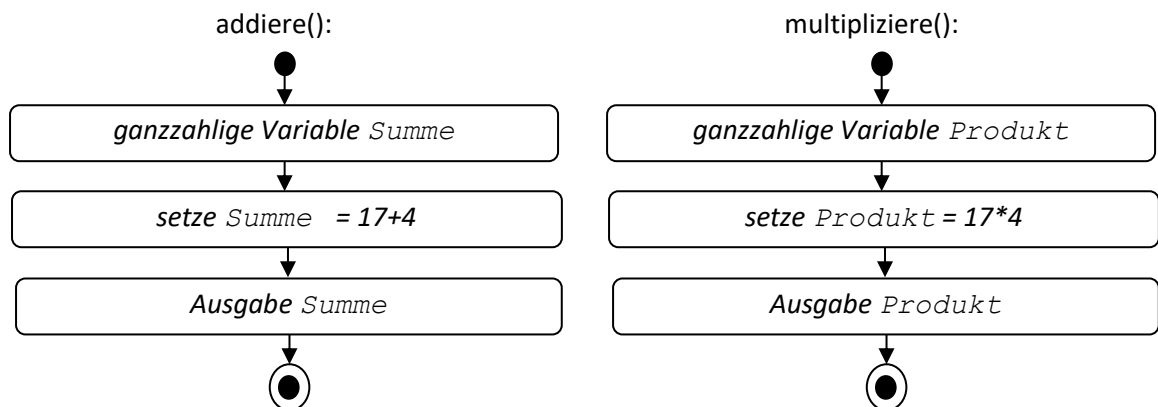


Während der Eingabe erhältst du in der Statuszeile Warn- und Fehlerhinweise

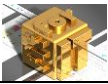
Mit den Schaltflächen unten kannst du zwischen der Konsole und den manchmal etwas ausführlicheren Fehlermeldungen hin- und herschalten.

6. In einer weiteren Funktion sollen zwei Zahlen multipliziert werden.

- Ergänze die Aktivitätsdiagramme für die Funktionen `addiere()` und `multipliziere()`.



- Codiere den Algorithmus (`06_rechnung_funktion`). (vgl. `.\262-materialien\rechnung\sketch_06_rechnung_funktion`)
7. Zusatzaufgabe: In einer weiteren Funktion soll der Quotient aus 17 und 4 berechnet und in der Konsole angezeigt werden (`07_division`). (vgl. `.\262-materialien\rechnung\sketch_07_division`)



Klassen

In EOS stehen eine ganze Reihe fertiger Klassen zur Verfügung, z. B. `Kreis` oder `Rechteck`. Bei objektorientierter Programmierung kann man auch eigene Klassen erstellen, was in EOS nicht geht. Für die Schreibweise in Processing wird der zusammenhängende Programmcode dargestellt:

Beispiel:

```
Addierer Addierer1;

void setup() {
  Addierer1=new Addierer();
  Addierer1.gibAus();
}

class Addierer {
  int Summe;
  Addierer() {
    Summe=17+4;
  }
  void gibAus () {
    println("17+4="+Summe);
  }
}
```

Objekt der Klasse Addierer mit dem Bezeichner "Addierer1".

Ein Objekt der Klasse benötigt einen Objektbezeichner und wird mit der Anweisung `new` erzeugt. Dabei wird der **Konstruktor** aufgerufen. Danach kann eine Methode des Objekts aufgerufen werden.

Eine Klasse wird mit dem Schlüsselwort `class` begonnen. Damit wird ein neuer Datentyp festgelegt, der wiederum Variablen und Funktionen beinhalten kann.

Die Variablen innerhalb einer Klasse werden als **Attribut** bezeichnet.

In dem **Konstruktor** werden die Attribute mit Default-Werten belegt.

Funktionen einer Klasse werden als **Methode** bezeichnet.

- Codiere die Klasse `Addierer` (`08_siebzehnplusvier_klasse`). (vgl. `.\262-materialien\rechnung\sketch_08_siebzehnplusvier_klasse`)
- Ergänze eine Klasse `Multiplizierer` (`09_rechnung_klasse`). Gib in dem Klassendiagramm rechts das Attribut und die Methode an. (vgl. `.\262-materialien\rechnung\sketch_09_rechnung_klasse`)

Klassendiagramm:

Multiplizierer
Produkt:int
<code>gibAus ()</code>

Attribute und Punktnotation

- Ändere das Programm `09_rechnung_klasse` so ab, dass in der Klasse `Multiplizierer` die zuvor berechnete Summe mit einer Zahl malgenommen wird. Du kannst auch die Vorlagendatei `sketch_v01_rechnung_klasse` verwenden. Speichere den Sketch unter `10_rechnung_Attribut`. (vgl. `.\262-materialien\rechnung\sketch_10_rechnung_Attribut`)

- Beschreibe die Meldung in der Statuszeile.

Eine Fehlermeldung wird angezeigt: *The variable „Summe“ does not exist“*

- Wird eine Variable innerhalb einer Klasse deklariert, handelt es sich um ein **Attribut**. Auf ein Attribut kann aber nur in Bezug auf ein Objekt zugegriffen werden.

- Teste den Zugriff auf das Attribut mit Hilfe der **Punktnotation** `Addierer1.Summe`. (`11_rechnung_Punktnotation`) (vgl. `.\262-materialien\rechnung\sketch_11_rechnung_Punktnotation`)

- Mit Hilfe der Punktnotation `Objekt.Attribut` bzw. `Objekt.Methode` kann auf ein Attribut bzw. auf eine Methode in Bezug auf ein Objekt zugegriffen werden.

- Zusatzaufgabe: Ergänze die Klasse `Dividierer`, in der das Produkt durch die Summe dividiert wird (`12_division`; siehe Klassendiagramm rechts). (vgl. `.\262-materialien\rechnung\sketch_12_division`)

Dividierer
Quotient:int
<code>gibAus ()</code>