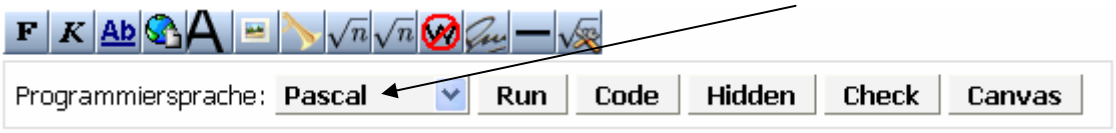
Vorbemerkungen

Die nachfolgenden Beispiele repräsentieren vielfältige Einsatzmöglichkeiten des PWikis in verschiedenen Programmierparadigmen und –sprachen.
Sie sind vollständig und können via copy-and-paste direkt in den Editor des PWikis übertragen werden. Achten Sie aber auf die korrekte Auswahl der Programmiersprache:



Gleichzeitig geben sie einen Einblick in ausgewählte Gestaltungselemente einer PWiki-Seite.

Beispiel: Summe natürlicher Zahlen (Pascal)

Quelltext:

```
<code>
function gausssumme(n: integer): double;
 var i    : integer;
     summe: double;
 begin
  summe:=0;
  for i:=1 to n do
   summe:=summe+i;
  result:=summe;
 end;
</code>

<check>
result:=(gausssumme(0)=0) and (gausssumme(100)=5050)
</check>

<run>
writeln(gausssumme(100));
</run>
```

Layout:

```
1  function gausssumme(n: integer): double;
2   var i    : integer;
3       summe: double;
4   begin
5    summe:=0;
6    for i:=1 to n do
7     summe:=summe+i;
8    result:=summe;
9   end;
10
```

Prima, deine Lösung scheint zu stimmen.

jetzt prüfen & speichern

```
1  writeln(gausssumme(100));
2
```

speichern & ausführen

```
>  5050.0
```

Beispiel: Turtlegrafik (Pascal)

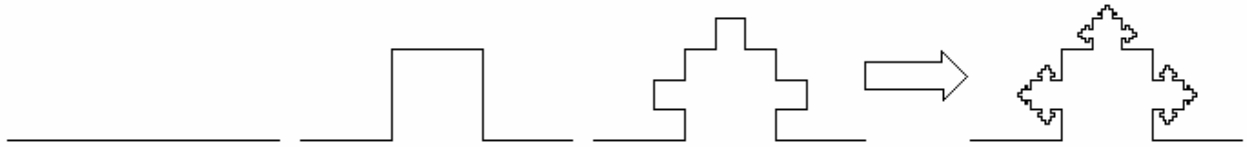
Quelltext:

```
== <font color=darkgreen size=4>''Quadratpflanze''</font> ==

Die Bilder zeigen die Entstehung einer Quadratpflanze.<br>
Bei jedem neuen Aufruf beträgt die Seitenlänge jeweils nur noch ein Drittel der
vorherigen.<br>
Weitere Informationen unter [http://www.inf-schule.de/
index.php?version=0&seite=informatik/rekursion/rekursionzahlen/uebungen].

[[Bild: Quadratpflanze1.GIF]]

'''Aufgaben:'''

# Beschreibe in der "Quadratpflanze" die Selbstähnlichkeiten.
#* Formuliere also die "grundlegenden Gedanken" für das Zeichnen der Pflanze.
#* Nach welcher Rekursionsvorschrift werden "Quadratpflanzen" erzeugt?
# Zeichne den nächsten (vierten) Schritt in der Entwicklung.
# Implementiere eine "Quadratpflanze" mit beliebiger Rekursionstiefe in der
Turtlegrafik.<br><br>

<code>
var laenge: real;

procedure pflanze(l: real; n: integer);
begin
 if n=1 then
  turtle_forward(l)
 else
  begin
   turtle_forward(l/3);
   turtle_left(90);
   pflanze(l/3,n-1);
   turtle_right(90);
   pflanze(l/3,n-1);
   turtle_right(90);
   pflanze(l/3,n-1);
   turtle_left(90);
   turtle_forward(l/3);
  end;
end;
</code>

<canvas w="600" h="400"></canvas>

<run>
selectcanvas(1);
canvas_clear();                    //Zeichenfläche löschen
laenge:=400;
turtle_move((600-laenge)/2, 300);  //Turtle ausrichten
turtle_right(90);
pflanze(laenge,5);
</run>
```

Layout:

Quadratpflanze [Bearbeiten]

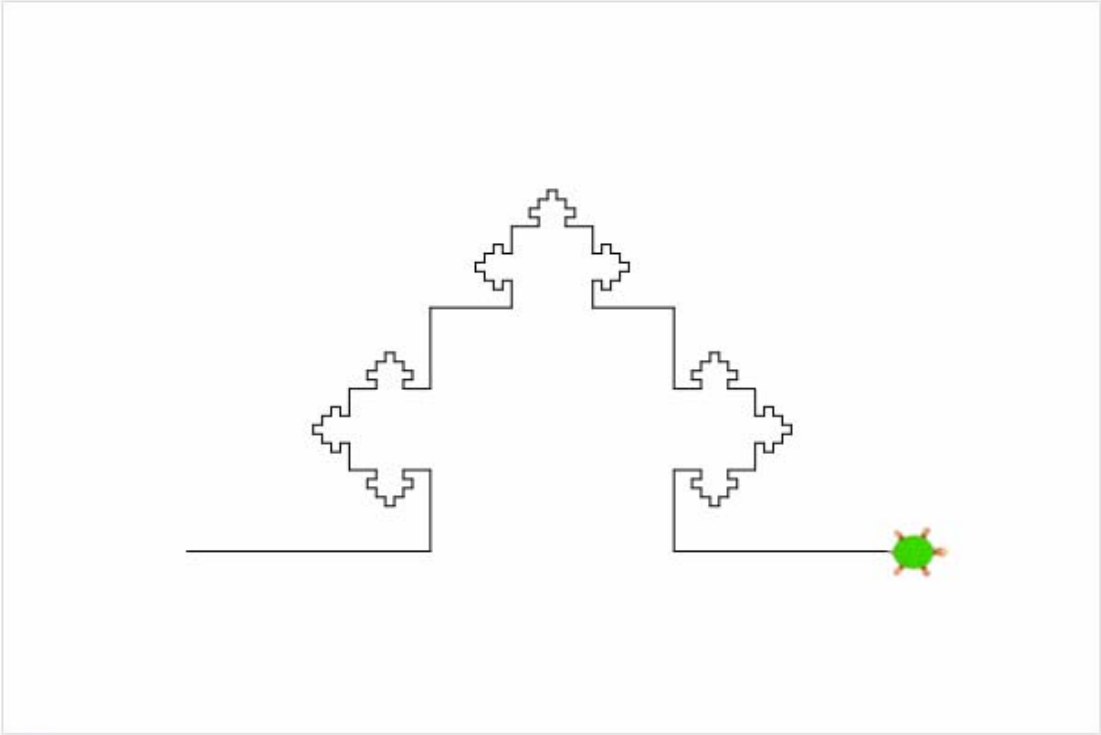
Die Bilder zeigen die Entstehung einer Quadratpflanze.
Bei jedem neuen Aufruf beträgt die Seitenlänge jeweils nur noch ein Drittel der vorherigen.
Weitere Informationen unter [1].



Aufgaben:

1. Beschreibe in der "Quadratpflanze" die Selbstähnlichkeiten.
   - Formuliere also die "grundlegenden Gedanken" für das Zeichnen der Pflanze.
   - Nach welcher Rekursionsvorschrift werden "Quadratpflanzen" erzeugt?
2. Zeichne den nächsten (vierten) Schritt in der Entwicklung.
3. Implementiere eine "Quadratpflanze" mit beliebiger Rekursionstiefe in der Turtlegrafik.

```
1  var laenge: real;
2
3  procedure pflanze(l: real; n: integer);
4  begin
5   if n=1 then
6    turtle_forward(l)
7   else
8    begin
9     turtle_forward(l/3);
10    turtle_left(90);
11    pflanze(l/3,n-1);
12    turtle_right(90);
13    pflanze(l/3,n-1);
14    turtle_right(90);
15    pflanze(l/3,n-1);
16    turtle_left(90);
17    turtle_forward(l/3);
18   end;
19 end;
20
```



```
1 selectcanvas(1);
2 canvas_clear();                    //Zeichenfläche löschen
3 laenge:=400;
4 turtle_move((600-laenge)/2, 300);  //Turtle ausrichten
5 turtle_right(90);
6 pflanze(laenge,5);
7
```

speichern & ausführen

```
>
```

Beispiel: Bubblesort (Java)

Quelltext:

```
== Bubblesort ==

<code>
public int[] ZufallsZahlen(int min, int max, int n) {
 int[] Zahlen = new int[n];
 for (int i = 0; i < n; i++)
  Zahlen[i] = (int)Math.floor(Math.random()*(max-min+1)+min);
 return Zahlen;
}

public void ZahlenAusgabe(int[] liste) {
 for (int i = 0; i < liste.length; i++)
  if (i < liste.length-1)
   System.out.print(liste[i]+", ");
  else
   System.out.println(liste[i]);
}

public void BubbleSort(int[] liste) {
 int h;
 for (int j = 0; j < liste.length-1; j++)
  for (int i = j; i < liste.length; i++)
   if (liste[i] < liste[j]) {
    h = liste[i]; liste[i] = liste[j]; liste[j] = h;
   }
}
</code>

<run>
int[] ZahlenListe = ZufallsZahlen(1, 1000, 10);
System.out.println("Unsortierte Zufallszahlen:");
ZahlenAusgabe(ZahlenListe);
System.out.println("Sortierte Zufallszahlen:");
BubbleSort(ZahlenListe);
ZahlenAusgabe(ZahlenListe);
</run>
```

Layout:

Bubblesort [Bearbeiten]

```
1  public int[] ZufallsZahlen(int min, int max, int n) {
2   int[] Zahlen = new int[n];
3   for (int i = 0; i < n; i++)
4    Zahlen[i] = (int)Math.floor(Math.random()*(max-min+1)+min);
5   return Zahlen;
6  }
7
8  public void ZahlenAusgabe(int[] liste) {
9   for (int i = 0; i < liste.length; i++)
10   if (i < liste.length-1)
11    System.out.print(liste[i]+", ");
12   else
13    System.out.println(liste[i]);
14 }
15
16 public void BubbleSort(int[] liste) {
17  int h;
18  for (int j = 0; j < liste.length-1; j++)
19   for (int i = j; i < liste.length; i++)
20    if (liste[i] < liste[j]) {
21     h = liste[i]; liste[i] = liste[j]; liste[j] = h;
22    }
23 }
24
```

```
1 int[] ZahlenListe = ZufallsZahlen(1, 1000, 10);
2 System.out.println("Unsortierte Zufallszahlen:");
3 ZahlenAusgabe(ZahlenListe);
4 System.out.println("Sortierte Zufallszahlen:");
5 BubbleSort(ZahlenListe);
6 ZahlenAusgabe(ZahlenListe);
7
```

speichern & ausführen

```
> Unsortierte Zufallszahlen:
358, 54, 161, 151, 126, 66, 905, 708, 564, 731
Sortierte Zufallszahlen:
54, 66, 126, 151, 161, 358, 564, 708, 731, 905
```

Beispiel: Objektorientierte Programmierung (Java)

Quelltext:

```
== Objektorientierte Programmierung ==

=== Technische Voraussetzungen ===

Zur Objektorientierten Programmierung in Java ist die Policy-Datei
* <tt>JavaPolicy.class</tt>
erforderlich. Entsprechende Hinweise findet man unter: [[JavaPolicy]]

=== Definition der Klasse ''Bruch''===

<code>
public class Bruch {

 protected int z;
 protected int n;

 public Bruch(int z, int n) {
  this.z = z;
  this.n = n;
 }

 public int Zaehler() {
  return z;
 }

 public int Nenner() {
  return n;
 }

 public void Ausgabe() {
  System.out.println(z + "/" + n);
 }
}
</code>

=== Instanz der Klasse ''Bruch''===

<run>
Bruch1 = new Bruch(1, 2);
System.out.print("Bruch1: "); Bruch1.Ausgabe();
</run>

=== Vererbung ===

<code>
public class GekuerztBruch extends Bruch {

 public GekuerztBruch(int z, int n) {
  super(z, n);
 }

 private int ggT(int m, int n) {
  if (n == 0)
   return m;
  else
   return ggT(n, m % n);
 }
```

```
 public void Kuerzen() {
  int teiler;
  teiler = ggT(z, n);
  z = z / teiler;
  n = n / teiler;
 }
}
</code>

<run>
Bruch2 = new GekuerztBruch(6, 8);
System.out.print("Bruch2, ungekürzt: "); Bruch2.Ausgabe();
Bruch2.Kuerzen();
System.out.print("Bruch2, gekürzt  : "); Bruch2.Ausgabe();
</run>
```

Layout:

Objektorientierte Programmierung [Bearbeiten]

Technische Voraussetzungen [Bearbeiten]

Zur Objektorientierten Programmierung in Java ist die Policy-Datei

- `JavaPolicy.class`

erforderlich. Entsprechende Installationshinweise findet man unter: JavaPolicy

Definition der Klasse *Bruch* [Bearbeiten]

```
1  public class Bruch {
2
3   protected int z;
4   protected int n;
5
6   public Bruch(int z, int n) {
7    this.z = z;
8    this.n = n;
9   }
10
11  public int Zaehler() {
12   return z;
13  }
14
15  public int Nenner() {
16   return n;
17  }
18
19  public void Ausgabe() {
20   System.out.println(z + "/" + n);
21  }
22 }
23
```

Instanz der Klasse *Bruch*

[Bearbeiten]

```
1 Bruch1 = new Bruch(1, 2);
2 System.out.print("Bruch1: "); Bruch1.Ausgabe();
3
```

speichern & ausführen

```
> Bruch1: 1/2
```

Vererbung

[Bearbeiten]

```
1  public class GekuerztBruch extends Bruch {
2
3   public GekuerztBruch(int z, int n) {
4    super(z, n);
5   }
6
7   private int ggT(int m, int n) {
8    if (n == 0)
9     return m;
10   else
11    return ggT(n, m % n);
12  }
13
14  public void Kuerzen() {
15   int teiler;
16   teiler = ggT(z, n);
17   z = z / teiler;
18   n = n / teiler;
19  }
20 }
21
```

```
1 Bruch2 = new GekuerztBruch(6, 8);
2 System.out.print("Bruch2, ungekürzt: "); Bruch2.Ausgabe();
3 Bruch2.Kuerzen();
4 System.out.print("Bruch2, gekürzt  : "); Bruch2.Ausgabe();
5
```

speichern & ausführen

```
> Bruch2, ungekürzt: 6/8
  Bruch2, gekürzt  : 3/4
```

Beispiel: Prozeduren höherer Ordnung (Scheme)

Quelltext:

```
== Differential- und Differenzenquotient ==

Aus der Mathematik ist die Definition des Differentialquotienten bekannt:

<blockquote>
<math>f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x)}{h}</math>
</blockquote>

Scheme kann natürlich nur mit einem endlichen Wert für h arbeiten.
Wir definieren deshalb:

<code>
(define h 1e-06)
</code>

Damit gehen wir vom Differential- zum Differenzenquotienten mit kleinen
Intervallen über.<br />
Nun können wir eine Prozedur höherer Ordnung definieren, die die ''Ableitung
beliebiger(!) Funktionen'' als Näherungswert ermittelt!

<code>
(define ableitung
  (lambda (fkt)
    (lambda (x)
      (/ (- (fkt (+ x h)) (fkt x)) h))))
</code>

Wir definieren eine Testfunktionen und rechnen nach.

<blockquote><math>f(x)=3x^2+\frac{2}{x}</math></blockquote>

<code>
(define fkt
  (lambda (x)
    ...))
</code>

<check>
(and (= (fkt -3) 79/3) (= (fkt 1) 5))
</check>

<run>
((ableitung fkt) 2)
</run>
```

Layout:

Differential- und Differenzenquotient [Bearbeiten]

Aus der Mathematik ist die Definition des Differentialquotienten bekannt:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Scheme kann natürlich nur mit einem endlichen Wert für h arbeiten. Wir definieren deshalb:

```
1 (define h 1e-06)
2
```

Damit gehen wir vom Differential- zum Differenzenquotienten mit kleinen Intervallen über.
Nun können wir eine Prozedur höherer Ordnung definieren, die die *Ableitung beliebiger(!) Funktionen* als Näherungswert ermittelt!

```
1 (define ableitung
2   (lambda (fkt)
3     (lambda (x)
4       (/ (- (fkt (+ x h)) (fkt x)) h))))
5
```

Wir definieren eine Testfunktionen und rechnen nach.

$$f(x) = 3x^2 + \frac{2}{x}$$

```
1 (define fkt
2  (lambda (x)
3    (+ (* 3 x x) (/ 2 x))))
4
```

Prima, deine Lösung scheint zu stimmen.

jetzt prüfen & speichern

```
1 ((ableitung fkt) 2)
2
```

speichern & ausführen

```
> 11.500003251896374
```

Beispiel: Vollständiger Graphendurchlauf (Prolog)

Quelltext:

```
== Vollständiger Graphendurchlauf ==

<table border="0" cellspacing="0" cellpadding="0">
<tr valign="top">
<td>

'''Problem:'''

Das bekannte Nikolaushaus soll in einem Zug gezeichnet werden, d.h. jede Kante
wird dabei genau einmal durchlaufen. Im Gegensatz zum Königsberger
Brückenproblem (Eulerzyklus) müssen dabei Start- und Zielknoten nicht
übereinstimmen.

Wie viele Möglichkeiten des vollständigen Durchlaufs dieses Graphen gibt es?

'''Lösungsidee:'''

Zunächst werden alle in Frage kommenden Strecken aus der Datenbasis
herausgesucht und in einer Streckenliste gesammelt. Das erste Element der
gewünschten Reiseroute (Knotenliste) soll der Startknoten des vollständigen
Graphendurchlaufs sein. Durch Tiefensuche werden nun die entsprechenden Kanten
ermittelt und dabei aus der Streckenliste herausgestrichen. Eine vollständiger
Durchlauf ist dann gefunden, wenn die vorgegebene Streckenliste leer ist.
</td>
<td>
[[Datei:Nikolaushaus1.GIF|miniatur|Das Haus des Nikolaus]]
</td>
</tr>
</table>

<popup name="Quelltext">
<code>
%Festlegung des Graphen (Nikolaushaus)
kante(a,b). kante(b,c). kante(c,d). kante(d,e). kante(e,a).
kante(b,d). kante(a,d). kante(e,b).

verbunden(A,B):-kante(A,B).
verbunden(A,B):-kante(B,A).

%Rundstrecke
rundstrecke(Route,Weg):-
      Route=[Anfang|_],
        findall([A,B],
            (kante(A,B),element(A,Route),element(B,Route)),
            Strecken),
      tsuche(Anfang,[Anfang],Weg,Strecken).

%Tiefensuche
tsuche(Knoten,Weg,Weg,[]):-!.
tsuche(Knoten,Teilpfad,Weg,Strecken):-
      verbunden(Knoten,NKnoten),
      teilliste([Knoten,NKnoten],Strecken),
      streichliste([Knoten,NKnoten],Strecken,NStrecken),
      tsuche(NKnoten,[NKnoten|Teilpfad],Weg,NStrecken).
```

```
%Suche nach allen möglichen Rundstrecken
suche(Route):-
      findall(Weg,rundstrecke(Route,Weg),Liste),
      ausgabe(Liste).

%Bildschirmausgabe
ausgabe(Liste):-ausgabe_h(Liste,0).
ausgabe_h([],Anzahl):-
        write('Es gibt '),write(Anzahl),write(' Moeglichkeit(en)!'),nl,!.
ausgabe_h([Weg|Liste],Anzahl):-
      ausgabe_l(Weg),nl,
      NAnzahl is Anzahl+1,
      ausgabe_h(Liste,NAnzahl).
ausgabe_l([X]):-write(X).
ausgabe_l([X|Liste]):-ausgabe_l(Liste),write(', '),write(X).

%Hilfsprädikate
streichliste([A,B],[[A,B]|L],L):-!.
streichliste([A,B],[[B,A]|L],L):-!.
streichliste(X,[Y|L1],[Y|L2]):-streichliste(X,L1,L2).

teilliste([A,B],[[A,B]|L]):-!.
teilliste([A,B],[[B,A]|L]):-!.
teilliste(X,[Y|L]):-teilliste(X,L).

element(X,[X|L]):-!.
element(X,[Y|L]):-element(X,L).
</code>
</popup>

<run>
suche([a,b,c,d,e]).
</run>
```

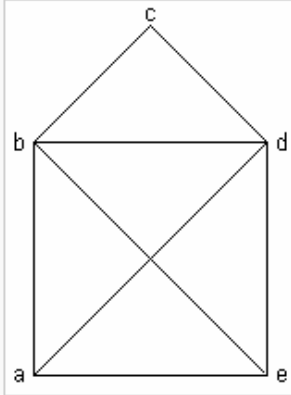
Layout:

Vollständiger Graphendurchlauf [Bearbeiten]

Problem:

Das bekannte Nikolaushaus soll in einem Zug gezeichnet werden, d.h. jede Kante wird dabei genau einmal durchlaufen. Im Gegensatz zum Königsberger Brückenproblem (Eulerzyklus) müssen dabei Start- und Zielknoten nicht übereinstimmen.

Wie viele Möglichkeiten des vollständigen Durchlaufs dieses Graphen gibt es?



Das Haus des Nikolaus

Lösungsidee:

Zunächst werden alle in Frage kommenden Strecken aus der Datenbasis herausgesucht und in einer Streckenliste gesammelt. Das erste Element der gewünschten Reiseroute (Knotenliste) soll der Startknoten des vollständigen Graphendurchlaufs sein. Durch Tiefensuche werden nun die entsprechenden Kanten ermittelt und dabei aus der Streckenliste herausgestrichen. Eine vollständiger Durchlauf ist dann gefunden, wenn die vorgegebene Streckenliste leer ist.

Quelltext anzeigen

```
1 suche([a,b,c,d,e]).
2
```

speichern & ausführen

Layout:

```
> suche([a,b,c,d,e])
a, b, c, d, e, a, d, b, e
a, b, c, d, e, b, d, a, e
a, b, c, d, b, e, a, d, e
a, b, c, d, b, e, d, a, e
a, b, c, d, a, e, b, d, e
a, b, c, d, a, e, d, b, e
a, b, d, e, a, d, c, b, e
a, b, d, e, b, c, d, a, e
a, b, d, c, b, e, a, d, e
a, b, d, c, b, e, d, a, e
a, b, d, a, e, b, c, d, e
a, b, d, a, e, d, c, b, e
a, b, e, a, d, c, b, d, e
a, b, e, a, d, b, c, d, e
a, b, e, d, c, b, d, a, e
a, b, e, d, b, c, d, a, e
a, d, e, a, b, c, d, b, e
a, d, e, a, b, d, c, b, e
a, d, e, b, c, d, b, a, e
a, d, e, b, d, c, b, a, e
a, d, c, b, d, e, a, b, e
a, d, c, b, d, e, b, a, e
a, d, c, b, a, e, b, d, e
a, d, c, b, a, e, d, b, e
a, d, c, b, e, a, b, d, e
a, d, c, b, e, d, b, a, e
a, d, b, c, d, e, a, b, e
a, d, b, c, d, e, b, a, e
a, d, b, a, e, b, c, d, e
a, d, b, a, e, d, c, b, e
a, d, b, e, a, b, c, d, e
a, d, b, e, d, c, b, a, e
a, e, b, c, d, b, a, d, e
a, e, b, c, d, a, b, d, e
a, e, b, d, c, b, a, d, e
a, e, b, d, a, b, c, d, e
a, e, b, a, d, c, b, d, e
a, e, b, a, d, b, c, d, e
a, e, d, c, b, d, a, b, e
a, e, d, c, b, a, d, b, e
a, e, d, b, c, d, a, b, e
a, e, d, b, a, d, c, b, e
a, e, d, a, b, c, d, b, e
a, e, d, a, b, d, c, b, e
Es gibt 44 Moeglichkeit(en)!
```

Beispiel: Prüfung (SQL)

Quelltext:

```
== Schüler/Prüfungsnoten ==

Gegeben ist folgende, stark vereinfachte Datenbank mit den Tabellen ''Schueler''
und ''Noten'', die neben den Adressen die Prüfungsnoten der Schüler in einigen
Fächern speichert.<br>

<popup label="Datenbank">
<code>

DROP TABLE IF EXISTS schueler;
CREATE TABLE schueler (
  Name varchar(20),
  Vorname varchar(20) ,
  Wohnort varchar(15) ,
  PLZ varchar(5) ,
  Telefon varchar(11) ,
  Straße varchar(25) ,
  Schuelernr varchar(7) ,
  PRIMARY KEY  (Schuelernr)
);

INSERT INTO schueler VALUES ('Scheurich', 'Erich', 'Bautzen', '02625', '40256',
'Mättigstr. 12', '1234567');
INSERT INTO schueler VALUES ('Schlenkrich', 'Ricarda', 'Teichritz', '02689',
'23456', 'Dorfstr. 2', '1234568');
INSERT INTO schueler VALUES ('Angst', 'Beatrice', 'Wurschen', '02645', '67255',
'Nr. 12a', '1234569');
INSERT INTO schueler VALUES ('Waurick', 'Andre', 'Bautzen', '02625', '30445',
'Brecht-Str. 34', '1234572');
INSERT INTO schueler VALUES ('Scheurich', 'Erick', 'Bautzen', '02625', '40256',
'Mättigstr. 12', '1234573');

DROP TABLE IF EXISTS Noten;
CREATE TABLE Noten (
  Schuelernr varchar(7),
  Mathe integer default '0',
  Deutsch integer,
  Bio integer
);

INSERT INTO Noten VALUES ('1234567', 2, 3, 1);
INSERT INTO Noten VALUES ('1234568', 2, 2, 2);
INSERT INTO Noten VALUES ('1234569', 3, 4, 2);
INSERT INTO Noten VALUES ('1234572', 4, 3, 3);
INSERT INTO Noten VALUES ('1234573', 1, 1, 1);

</code>
</popup>

'''Erstellen Sie folgende Abfragen:'''

# Gesucht sind alle Namen, Vornamen und Adressen derjenigen Schüler, deren Noten
in allen Fächern besser als 3 sind.
# Gesucht sind alle Schüler aus Bautzen.

<run>
</run>
```

Layout:

Schüler/Prüfungsnoten [Bearbeiten]

Gegeben ist folgende, stark vereinfachte Datenbank mit den Tabellen *Schueler* und *Noten*, die neben den Adressen die Prüfungsnoten der Schüler in einigen Fächern speichert.

Datenbank anzeigen

Erstellen Sie folgende Abfragen:

1. Gesucht sind alle Namen und Vornamen der Schüler, deren Noten in allen Fächern besser als 3 sind.
2. Gesucht sind alle Schüler aus Bautzen.

```
1  select name, vorname, mathe, deutsch, bio
2  from schueler, noten
3  where schueler.schuelernr = noten.schuelernr
4        and mathe < 3 and deutsch < 3 and bio < 3;
5
6  select *
7  from schueler
8  where wohnort like 'Bautz%';
9
```

speichern & ausführen

```
> +-------------+---------+-------+---------+-----+
  | NAME        | VORNAME | MATHE | DEUTSCH | BIO |
  +-------------+---------+-------+---------+-----+
  | Schlenkrich | Ricarda | 2     | 2       | 2   |
  | Scheurich   | Erick   | 1     | 1       | 1   |
  +-------------+---------+-------+---------+-----+
  2 rows in set.

  +-----------+---------+----------+-------+---------+-----------------+------------+
  | NAME      | VORNAME | WOHNORT  | PLZ   | TELEFON | STRASSE        | SCHUELERNR |
  +-----------+---------+----------+-------+---------+-----------------+------------+
  | Scheurich | Erich   | Bautzen  | 02625 | 40256   | Mättigstr. 12  | 1234567    |
  | Waurick   | Andre   | Bautzen  | 02625 | 30445   | Brecht-Str. 34 | 1234572    |
  | Scheurich | Erick   | Bautzen  | 02625 | 40256   | Mättigstr. 12  | 1234573    |
  +-----------+---------+----------+-------+---------+-----------------+------------+
  3 rows in set.
```