# Inhaltsverzeichnis

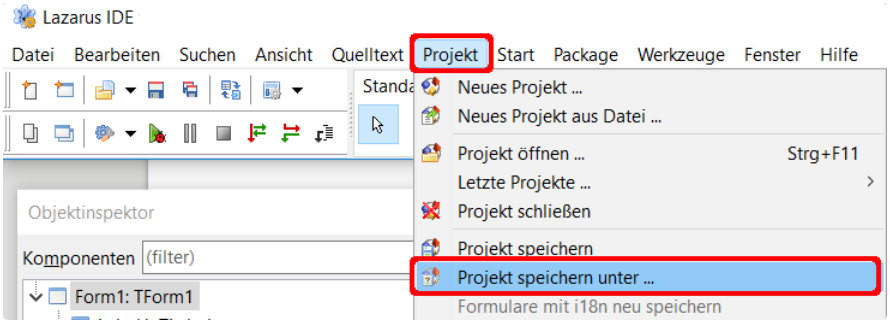
# Das erste Projekt

## Umrechnung Celsius → Fahrenheit

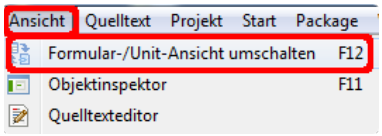
Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Lege zunächst einen Ordner an, in dem du die Projektdateien speicherst.

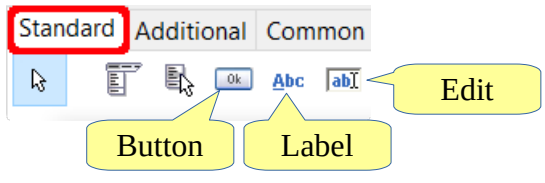
2 Speichere das Projekt in dem neu erstellten Ordner.

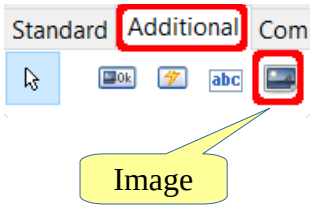


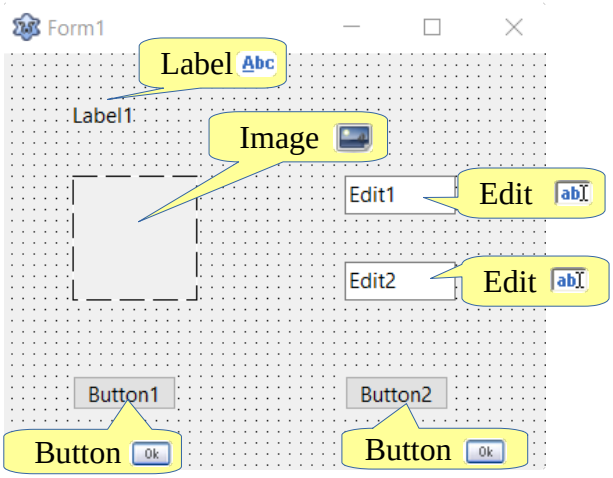

3 Erstelle im Formdesigner die grafischen Komponenten.
Sollte der Formdesigner nicht sichtbar sein:



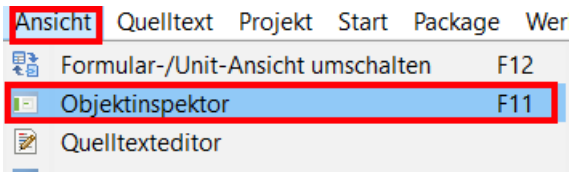

Unter dem Reiter „Standard“ findest du das Label, die Buttons und das Textfeld.
Im Reiter „Additional“ befindet sich das Icon für das Einfügen eines Bilds.
Das Bild des Thermometers wird später eingefügt.



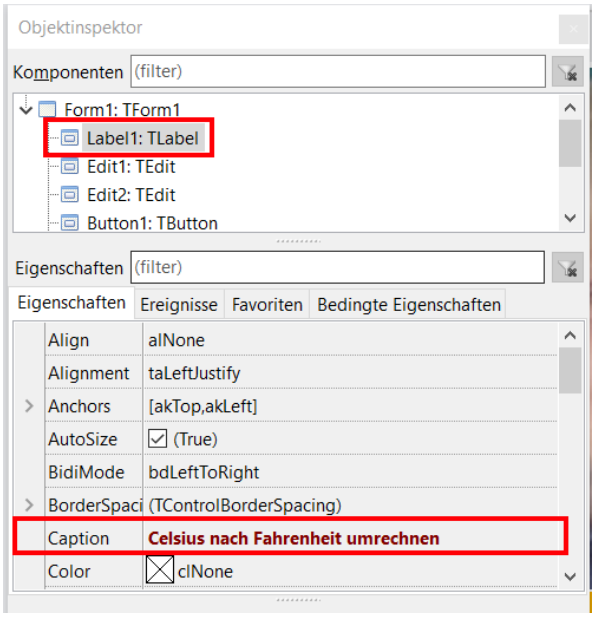



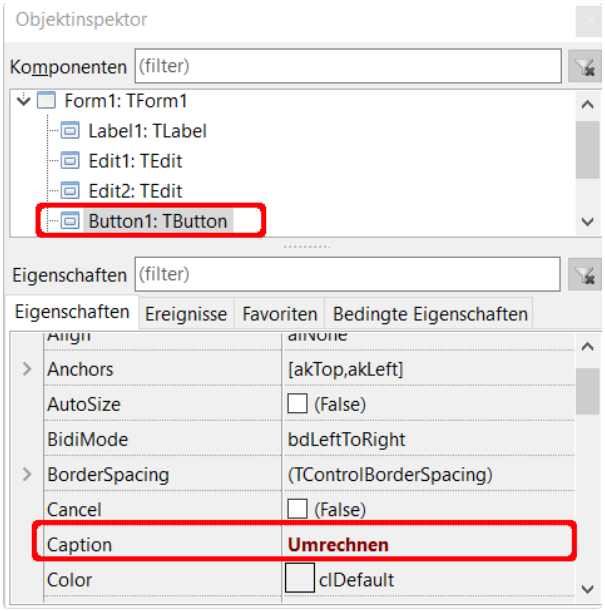



4 Im Objektinspektor musst du die Texte der Buttons und des Labels verändern.

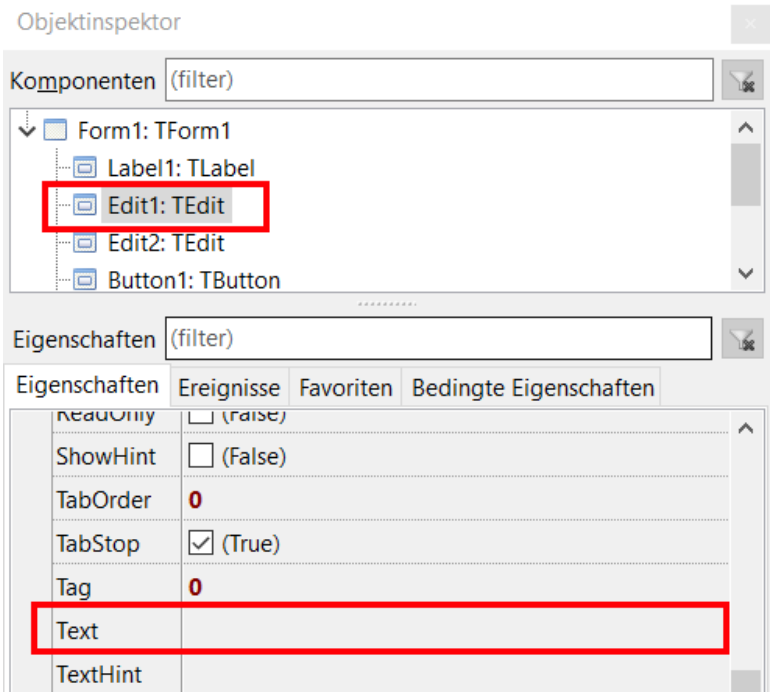



Beim Label und den Buttons heißt die Eigenschaft „Caption“

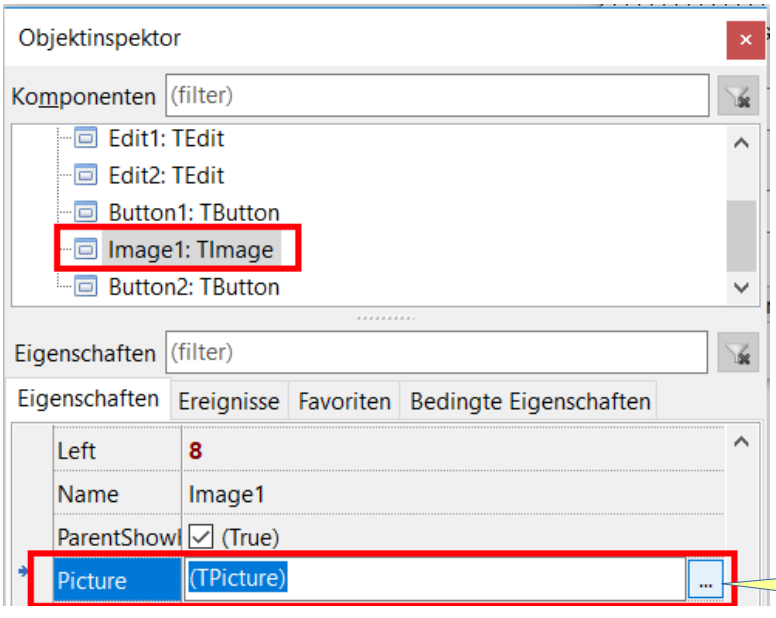


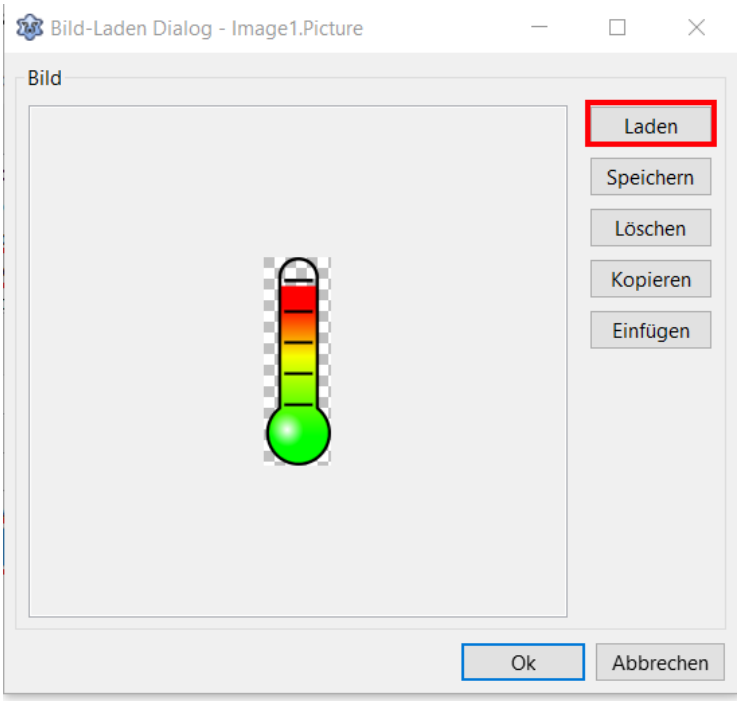



Beim Textfeld (Edit1) musst du die Eigenschaft „Text“ suchen. Der Text soll leer bleiben.




Zum Schluss muss noch das Bild zugewiesen werden:

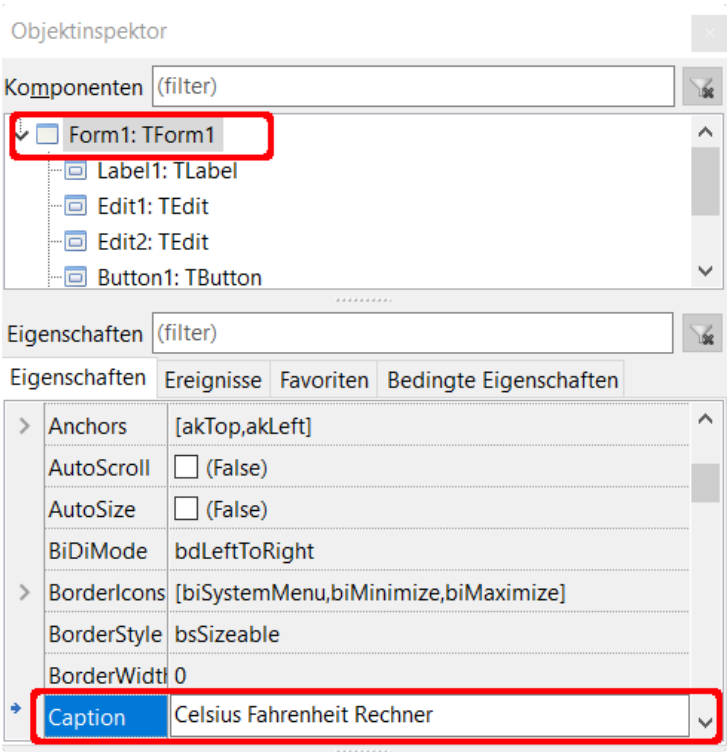




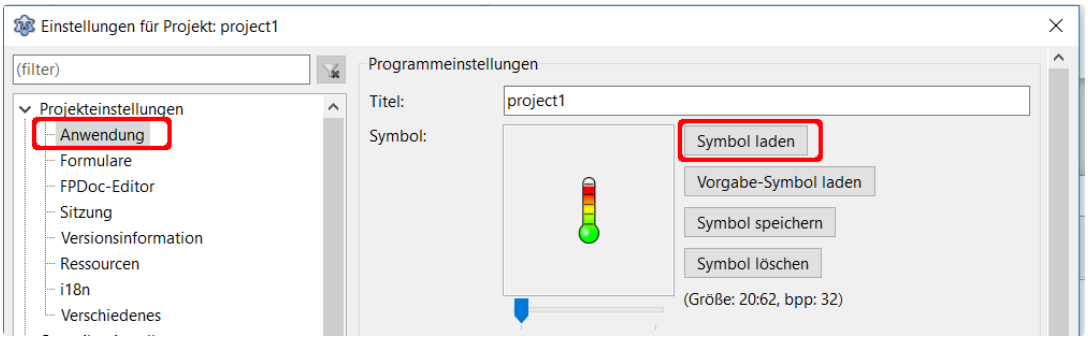

Die Grafik zum Thermometer findest du hier:

Anschließend musst du noch bei den Eigenschaften einen Haken bei „AutoSize“ setzen.

Als Nächstes musst du den Text in der Titelzeile ändern:

Als Letztes kannst du noch das Symbol in der Titelzeile verändern. Unter Projekt → Projekteinstellungen legst du das Symbol fest. Das Symbol findest du hier.




5 Jetzt kannst du mit der Programmierung beginnen.
Klicke doppelt auf den Button „Beenden“.
Im Quelltexteditor wird eine Prozedur erstellt:

```
procedure TForm1.Button2Click(Sender: TObject);
begin

end;
```

Prozeduren sind „Unterprogramme“, sie haben die Aufgabe, häufig wiederkehrende Befehle zusammenzufassen. Jede Prozedur erhält einen eindeutigen Namen, über den sie ausgeführt werden kann.
Jede Prozedur besteht aus dem Schlüsselwort `procedure`, gefolgt von einem gültigen Namen und eventuell einer Liste von Variablen in runden Klammern. Diesen Teil nennt man Kopf der Prozedur. Danach können Variable definiert werden und anschließend zwischen `begin` und `end`; die Anweisungen, die die Prozedur ausführen soll.
Sender weist auf das Objekt hin, von dem das Ereignis (Click) ausgelöst wurde, in diesem Fall der Button2.

Setze zwischen begin und end; den Befehl
`Application.terminate;`
ein. Damit wird das Programm beendet.

6 Nach einem Doppelklick auf den Button „Berechnen“ wird die entsprechende Prozedur im Quelltexteditor erstellt.

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Als Erstes brauchst du eine Variable, mit der die Eingabe aus dem oberen Eingabefeld (Celsius) nach Fahrenheit umgerechnet wird und im zweiten Eingabefeld angezeigt wird.

Eine Variable kann Daten innerhalb eines Programms speichern. Sie besteht immer aus einem Namen und einem Typ.

Es gibt verschiedene Datentypen, hier die wichtigsten:

| | |
|---|---|
| Strings | → Zeichenketten |
| Char | → einzelnes Zeichen |
| Integer | → ganze positive und negative Zeilen |
| Fließkommazahlen | → Double |

Die Variable soll nur in der Prozedur gültig sein, ist also eine lokale Variable. Da auch Kommazahlen eingegeben werden können, muss der Datentyp Double sein.
**Die Deklaration erfolgt am Anfang der Prozedur vor dem begin.**
Mit var wird die Deklaration eingeleitet, dann folgt der Name der Variable und hinter dem Doppelpunkt der Typ der Variable.

**Beispiele:**
```
Text: String;
Buchstabe: Char;
Nummer: Integer;
Kommazahl: Double;
```

**Beim Namen der Variable sind keine Umlaute oder Leerstellen erlaubt.**

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Fahrenheit: Double;
```

Der Wert von Textfeldern ist immer eine Variable vom Typ String. Sie muss in eine Zahl vom Typ Double umgewandelt werden (Typecast). Das erledigt der Befehl `StrToFloat`.

```
begin
    Fahrenheit:= StrToFloat(Edit1.Text);
```

Mit `:=` wird der Variablen ein Wert zugewiesen.
Anschließend wird die Temperatur in Fahrenheit berechnet …

```
Fahrenheit:=  Fahrenheit * 9/5 + 32;
```

… und zum Schluss im unteren Textfeld der Wert in Fahrenheit ausgegeben. Jetzt muss die Zahl mit `FloatToStr` in einen String zurück verwandelt werden.

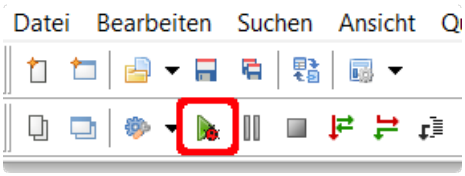
```
Edit2.Text:= FloatToStr(Fahrenheit);
```

Etwas unschön ist noch, dass die angezeigte Temperatur in Fahrenheit mehr als 1 Nachkommastelle haben kann. Mit zwei Ergänzungen kannst du das Problem lösen:
Füge unter `uses` die Bibliothek `Math` hinzu.

Bevor die Temperatur in Fahrenheit in Edit2 angezeigt wird, wird nach der ersten Nachkommastelle gerundet.

```
Fahrenheit:= RoundTo(Fahrenheit,-1);
```

7

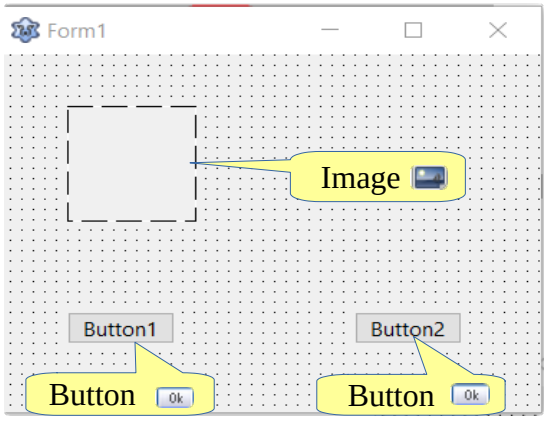



Das Programm ist jetzt startbereit:

# Wenn → Dann Entscheidungen

## Würfelsimulation

1




Die Form enthält zwei Buttons und ein Bild. Ändere im Objektinspektor die Beschriftung der Buttons und die Statuszeile.

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen. Um es ausprobieren zu können, musst du die Bilder (1.png … 6.png) im gleichen Ordner wie das Programm speichern

2

Ein Doppelklick auf den Button „Beenden“ erstellt die Prozedur
TForm1.Button1Click(Sender: Tobject);
Setze zwischen Begin und end; den Befehl

```
Application.Terminate;
```

ein.

3

Erstelle mit einem Doppelklick auf den Button „Würfeln“ die Prozedur
`TForm1.Button1Click(Sender: Tobject);`

Als Erstes brauchst du eine Variable vom Typ Integer.
Außerdem musst du den „Zufallszahlgenerator“ (Randomize) starten.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   Zahl: Integer;
begin
   Randomize;
```

Nach dem Start des „Zufallszahlgenerators“ wird eine Zahl gewürfelt:

```
   Zahl:= Random(6) + 1;
```

Es wird jetzt eine Zufallszahl erstellt, die größer gleich 0 und kleiner 6 ist. Da die 0 natürlich nicht vorkommen darf und die 6 vorkommen soll, wird immer 1 addiert.

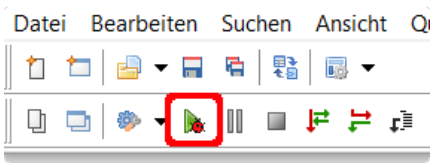
4 Jetzt fehlt noch die Abfrage, welcher Würfel angezeigt werden soll, wenn die Zahl gewürfelt wurde.
Das erledigt eine if-Abfrage:

```
   if Zahl = 1 then Image1.Picture.LoadFromFile('1.png');
   if Zahl = 2 then Image1.Picture.LoadFromFile('2.png');
```

Wenn (if) Zahl den Wert 1 hat, dann (then) lade die Datei 1.png.

Du musst jetzt nur noch für die anderen möglichen Ergebnisse die if-Abfragen ergänzen.

Die Bilder der Würfel findest du hier. Du musst sie herunterladen und im Projektordner speichern.

5 

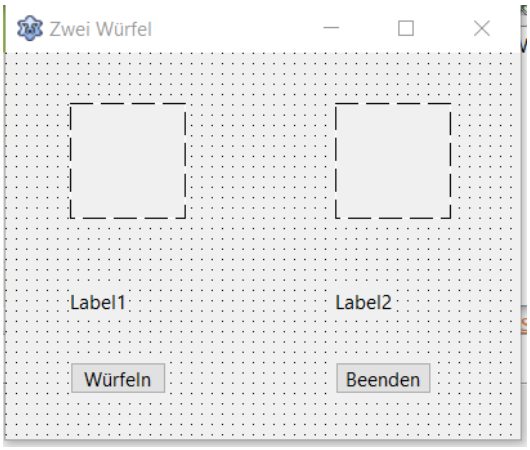

Das Programm ist jetzt startbereit.


Statt komplexer if-Abfragen versuche doch mal eine case-Abfrage:

```
case Zahl of
   1: Image1.Picture.LoadFromFile('1.png');
   2: Image1.Picture.LoadFromFile('2.png');
   3: Image1.Picture.LoadFromFile('3.png');
   4: Image1.Picture.LoadFromFile('4.png');
   5: Image1.Picture.LoadFromFile('5.png');
   6: Image1.Picture.LoadFromFile('6.png');
end;
```

# Würfelsimulation mit zwei Würfeln

1 Erweitere die Form um ein weiteres Bild und zwei Labels.




Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

2 Bei der Prozedur Button1Click brauchst du jetzt zwei Variable vom Typ Integer ...

```
var
  Zahl1, Zahl2: Integer;
```

… und für jeden Wurf eine Zufallszahl:

```
Zahl1:= Random(6) + 1;
Zahl2:= Random(6) + 1;
```

3 Verwende jetzt entweder die if- oder die case-Abfrage, um den gewürfelten Zahlen das entsprechende Bild zuzuweisen
(Zahl1 → Image1, Zahl2 → Image2 ...)

4 Lösche in den beiden Labels den vorgegebenen Text, es sieht beim Programmstart unschön aus.

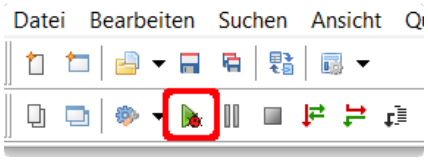
5 In Label1 soll die Gesamtzahl der Augen angezeigt werden:

```
Label1.Caption:='Gesamtzahl der Augen: ' + IntToStr(Zahl1 + Zahl2);
```

Die Eigenschaft „Caption“ wird immer als Text (String) dargestellt. Bei der Summe von Zahl1 und Zahl2 handelt es sich aber um eine Variable vom Typ Integer. Sie muss in einen String umgewandelt werden (Typecast). Das erledigt `IntToStr`. Das +-Zeichen verbindet die beiden Texte (Strings) miteinander.

6 Label2 soll anzeigen, dass ein Pasch (beide Würfel zeigen die gleiche Augenzahl) gewürfelt wurde. Das Problem ist nur, wenn auf einen Pasch die Würfel unterschiedliche Ergebnisse zeigen, das Label2 immer noch den Pasch anzeigt.
Eine Erweiterung der if-Abfrage mit else schafft hier Abhilfe.
Wenn (`if`) Zahl1 gleich Zahl2 dann (`then`) zeigt Label2.Caption ‘Pasch!‘ ansonsten (`else`) bleibt Label2.Caption leer.

```
if Zahl1 = Zahl2 then Label2.Caption:= 'Pasch!' else Label2.Caption:= '';
```

7 

Starte jetzt das Programm.

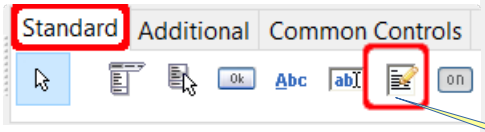
# Schleifen mit for und while

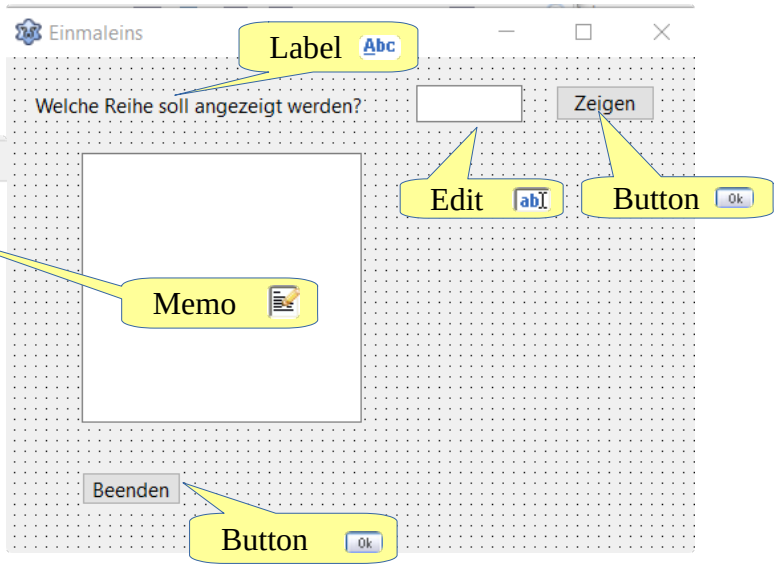
## Einmaleinsreihen mit for

Nach Eingabe in einem Textfeld sollen Einmaleinsreihen angezeigt werden.

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

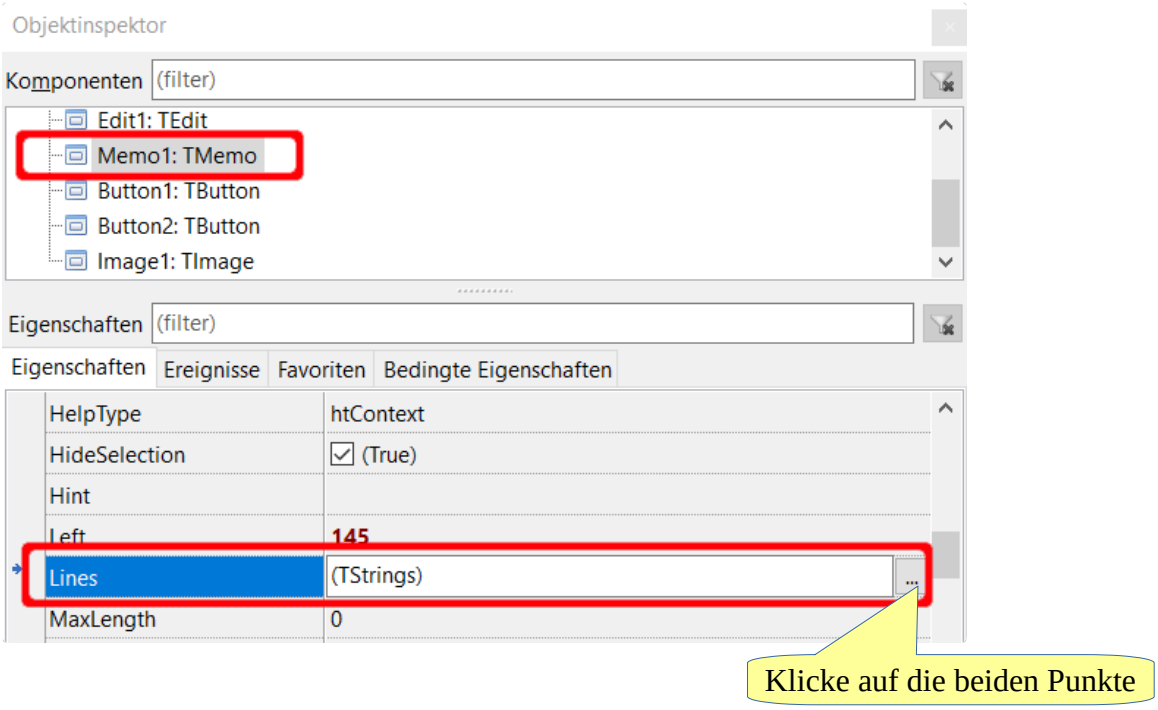
1 Erstelle die Form mit einem Label, einem Edit-Feld, zwei Buttons und einem Memo.

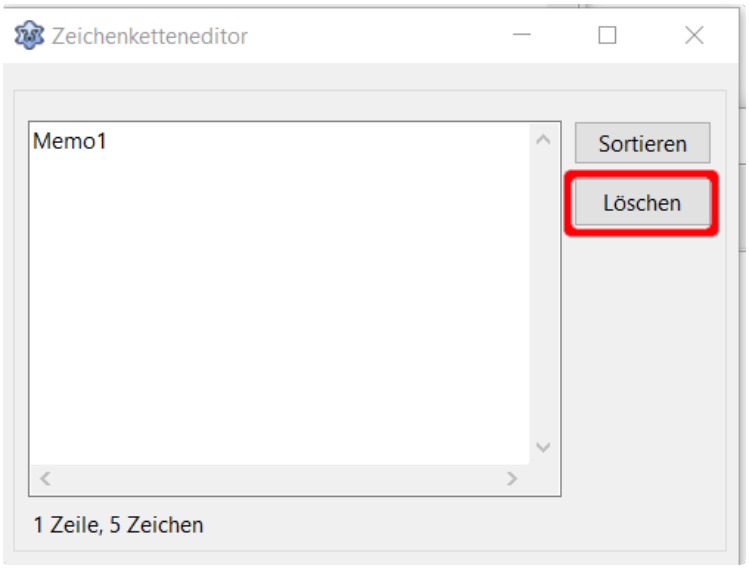


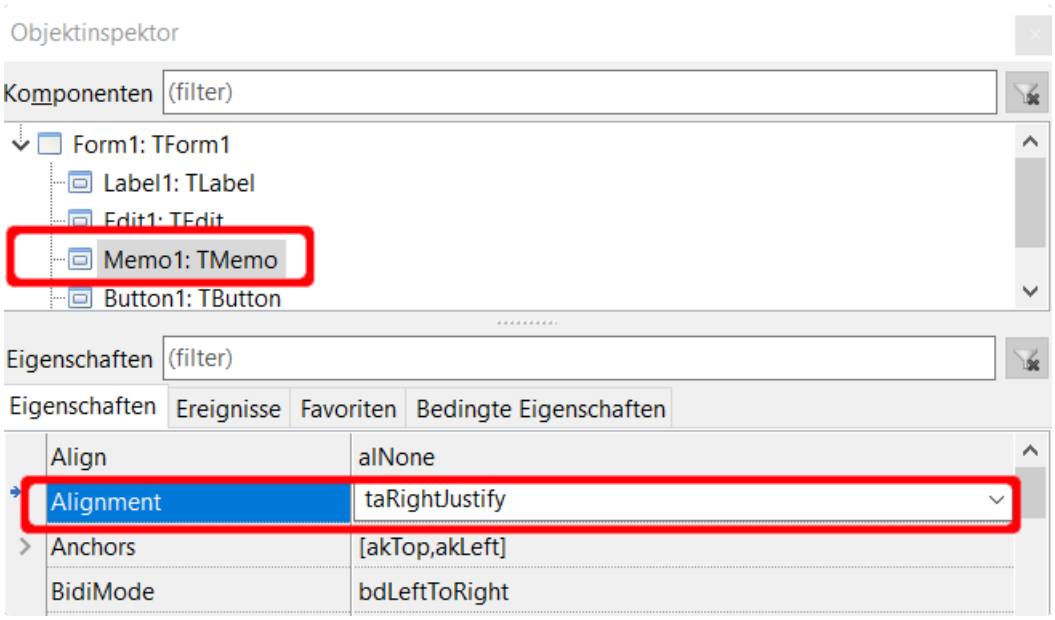



2 Verändere die Texte für die Buttons(Button1.Caption, Button2.Caption), leere das Textfeld (Edit1.Text) und ändere die Statuszeile.
Das Icon findest du hier.

3 Beim Memo-Feld musst du zwei Eigenschaften ändern:






3 Jetzt beginnst du mit dem eigentlichen Programm. Klicke doppelt auf den Button „Zeigen“. Du brauchst zwei Variable vom Typ Integer: eine als Schleifenvariable und eine Variable in der das jeweilige Ergebnis gespeichert wird.

```
procedure TForm1.Button1Click(Sender: TObject);

var

   Zaehler, Ergebnis: Integer;
```

4 Das Durchzählen von 1-mal bis 10-mal der eingegebenen Zahl erledigt eine `for`-Schleife. Sie hat die Form:

**von (for) Startwert:= Start bis Endwert mache (do)**

in diesem Fall:

```
for Zaehler:= 1 to 10 do

begin
```

Ergebnis berechnen:

```
Ergebnis:= Zaehler * StrToInt(Edit1.Text);
```

Weil das Edit-Feld Text enthält, muss es zur Berechnung in eine Variable vom Type Integer umgewandelt (`StrToInt`) werden.

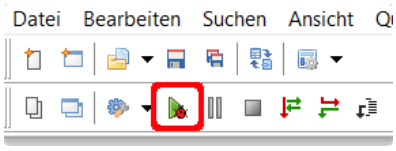
5 Zum Schluss wird das Ergebnis jeweils als neue Zeile dem Memo-Feld hinzugefügt. Es muss aber vorher mit IntToStr in einen String zurück verwandelt werden.

```
Memo1.Lines.add(IntToStr(Ergebnis));
```

6 Wenn mehrere Reihen hintereinander angezeigt werden, bleiben die vorherigen Reihen im Memo erhalten. Das kannst durch Einfügen von ...

```
Memo1.clear;
```

… unmittelbar vor der for-Schleife verhindern.

7 


Starte jetzt das Programm.

## Einmaleinsreihen mit while

Du kannst auch while als Schleife einsetzen. While ist eine „kopfgesteuerte“ Schleife, im Schleifenkopf wird entschieden, ob noch ein weiterer Durchlauf gestartet wird.
Hierzu sind ein paar kleine Änderungen nötig:

1 Der Startwert der Schleife wird durch Zuweisung der Variable Zaehler festgelegt:

```
Zaehler:= 1;
```

2 Im Schleifenkopf wird der Endwert (kleiner oder gleich 10) festgelegt.

Solange (`while`) Zaehler kleiner oder gleich 10 mache (`do`)

```
while Zaehler <= 10 do
begin
   Ergebnis:= Zaehler * StrToInt(Edit1.Text);
   Memo1.Lines.add(IntToStr(Ergebnis));
```

Du musst dich im Unterschied zur for-Schleife selbst um die Erhöhung der Zählervariable kümmern. (inc = increase → erhöhen)

```
  inc(Zaehler);
end;
```

**Schleife mit repeat**

Das Programm mit der „fußgesteuerten“ Schleife repeat:

```
repeat
   Ergebnis:= Zaehler * StrToInt(Edit1.Text);
   Memo1.Lines.add(IntToStr(Ergebnis));
   inc(Zaehler);
until Zaehler > 10;
```

Es wird erst am Ende überprüft, ob der Wert der Variable Zaehler überschritten ist. Das ist erst dann der Fall, wenn sie > 10 ist.

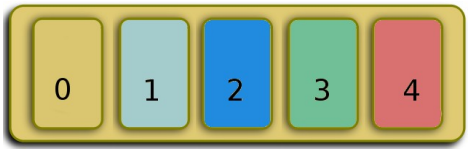
**Zwei Probleme gibt es noch:**

1. Der Wertebereich von Integer-Variablen ist auf 2.147.483.647 begrenzt. Es können also nicht beliebig große Einmaleinsreihen angezeigt werden.
2. Wenn das Textfeld leer bleibt, wird eine Fehlermeldung erzeugt. Wenn du das verhindern willst, musst du vor der Schleife eine Bedingung einfügen, die überprüft, ob das Eingabefeld nicht leer (`<>''`) ist:
   ```
   if Edit1.Text<>'' then
   ```

**Denke an begin und end;**

# Arrays – Datenfelder

## Datenbank mit einer Listbox




Du kannst dir ein Array wie einen Schrank mit Schubladen vorstellen. In jeder Schublade befindet sich ein Element. Mit Hilfe von Arrays können Daten eines einheitlichen Typs (z. B. String, Integer) im Speicher abgelegt und jederzeit wieder hervor geholt werden.

Inhalt von Schublade Nummer 1 → Schublade[1]
Inhalt von Schublade Nummer 2 → Schublade[2]
Ein Array wird so definiert:

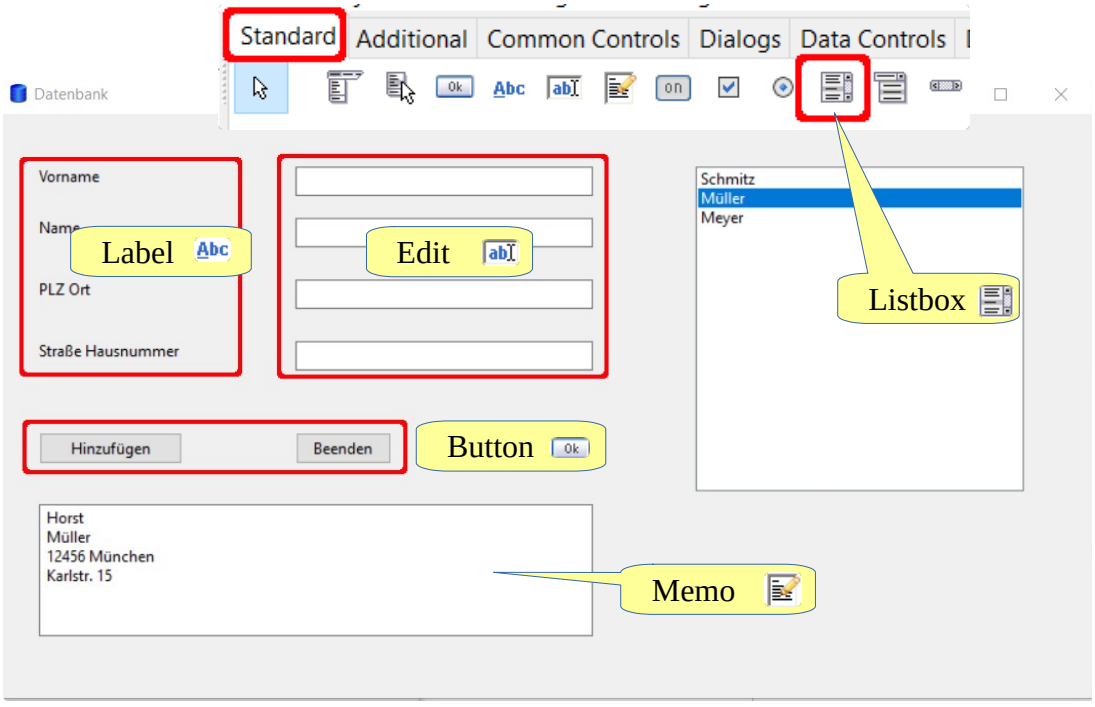
**Name des Arrays: Array [1. Element .. letztes Element] of Datentyp;**

```
Vorname: Array [0..100] of String;
```

Ein Überschreiten der Anzahl der Elemente zur Laufzeit des Programms führt zu einer Fehlermeldung. Am Ende des Kapitels erfährst du, wie man dynamische Arrays erstellt.

Im Programm sollen die Daten erfasst werden. Ein Klick auf den Button „Hinzufügen“ schreibtt die Daten in ein Array und fügt den Nachnamen in der Listbox hinzu. Ein Kick in der ListBox auf den Namen macht den vollständigen Datensatz im Memo-Feld sichtbar.

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.



1 Erstelle die Form. Hier findest du das Symbol für die Statuszeile.
Gib den Edit-Feldern im Objektinspektor unter Eigenschaften → Name aussagekräftige Namen. In den Erläuterungen werden diese Namen verwendet.

Edit1 → EditVorname

Edit2 → EditName

Edit3 → EditOrt

Edit4 → EditStrasse

2 Als Erstes musst du unter `var` die Arrays definieren:

```
var
  Vorname: Array [0..100] of String;
  Nachname: Array [0..100] of String;
  Ort: Array [0..100] of String ;
  Strasse: Array [0..100] of String;
```

3 Klicke doppelt auf den Button „Hinzufügen“.
In der Prozedur wird jeweils den Elementen der Arrays der Inhalt des entsprechenden Textfeldes zugewiesen.

```
Vorname[Zaehler]:= EditVorname.Text;
Nachname[Zaehler]:= EditName.Text;
Ort[Zaehler]:= EditOrt.Text;
Strasse[Zaehler]:= EditStrasse.Text;
```

4 Anschließend wird der Listbox das Element, das den Nachnamen enthält, hinzugefügt ...

```
ListBox1.Items.Add(Nachname[Zaehler]);
```

… und die Variable Zaehler um 1 erhöht ...

```
inc(Zaehler);
```

… und die Inhalte der Textfelder gelöscht.

```
EditVorname.Text:= '';
EditName.Text:= '';
. . .
```

5 Nach einem Doppelklick in die ListBox wird eine neue Prozedur angelegt

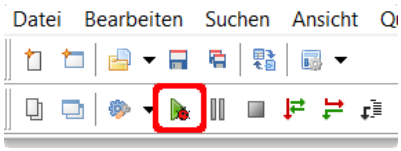
```
procedure TForm1.ListBox1Click(Sender: Tobject);
```

Als Erstes löschst du den Inhalt des Memo-Felds …

```
Memo1.Clear;
```

… und fügst (`add`) für jedes Array das aktuelle Element (`ListBox1.ItemIndex`) hinzu.

```
Memo1.Lines.add(Vorname[ListBox1.ItemIndex]);
Memo1.Lines.add(Nachname[ListBox1.ItemIndex]);
. . .
```

6 


Fertig, du kannst das Programm starten.


## Dynamische Arrays

### Dynamisches Array definieren

```
Vorname: Array of String;
Nachname: Array  of String;
Ort: Array  of String;
Strasse: Array of String;
```

**Vor einer neuen Zuweisung wird die Anzahl der Elemente des Arrays mit `SetLength` jeweils um 1 erhöht:**

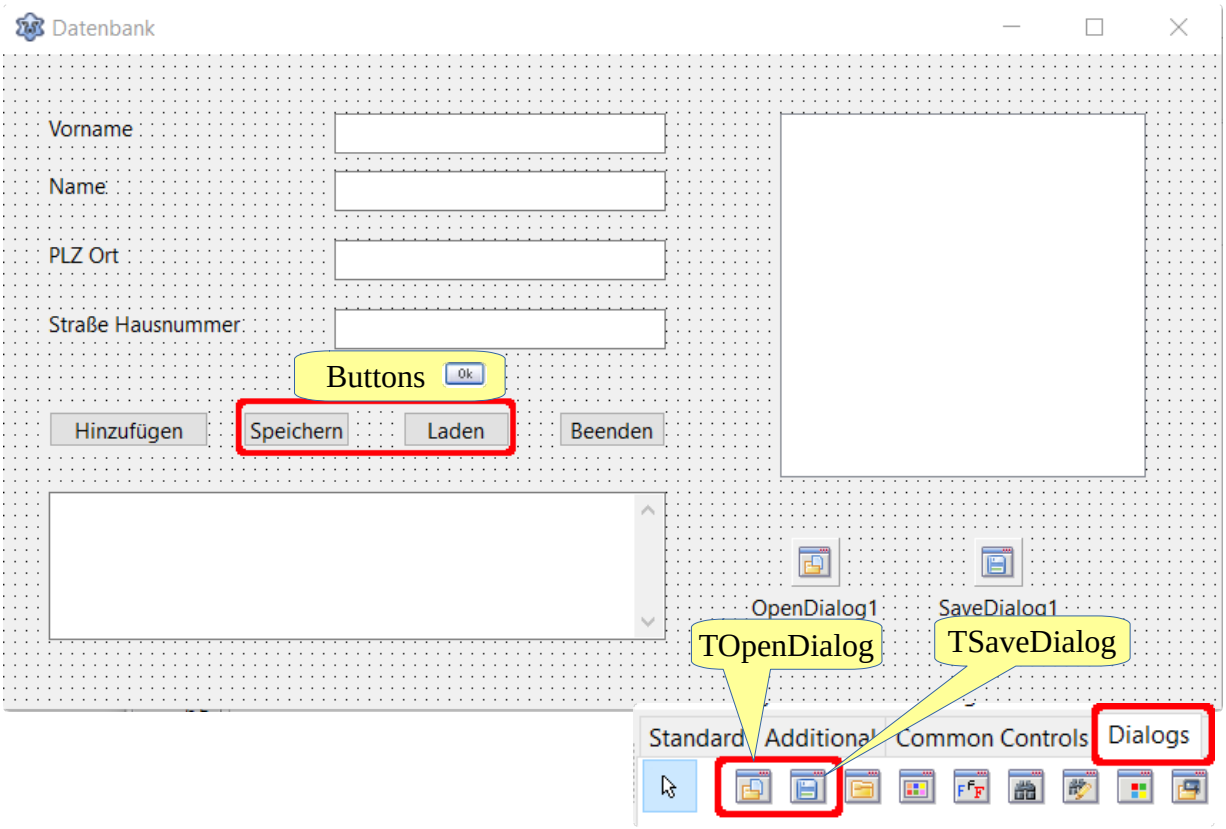
```
SetLength(Vorname, Zaehler + 1);
SetLength(Nachname, Zaehler + 1);
SetLength(Ort, Zaehler + 1);
SetLength(Strasse, Zaehler + 1);
Vorname[Zaehler]:= EditVorname.Text;
Nachname[Zaehler]:= EditName.Text;
Ort[Zaehler]:= EditOrt.Text;
Strasse[Zaehler]:=EditStrasse.Text;
```

# Daten laden und speichern

Es wäre natürlich schön, wenn man die Daten, die im Programm erhoben wurden, auch speichern und wieder laden könnte. Das Einbinden dieser Funktionen ist mit etwas Aufwand verbunden.

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen. Eine Beispieldatei für die Daten gibt es hier (alle Daten sind natürlich frei erfunden).

1 Füge als Erstes zwei zusätzliche Buttons hinzu. Beschrifte sie mit „Speichern“ und „Laden“




2 Im Reiter Dialogs befinden sich die Dialoge zum Öffnen (TOpenDialog) und Speichern (TSaveDialog) einer Datei. Du kannst sie an beliebiger Stelle platzieren: Sie werden zur Laufzeit des Programms nicht angezeigt.

3 Du kannst im Objektinspektor noch einige Eigenschaften des OpenDialog1 verändern: DefaultExt legt die Erweiterung des Dateinamens fest. Bei der Wahl des Namens zur Laufzeit des Programms muss dann keine Erweiterung angegeben werden.
Die Angabe eines Filters sorgt dafür, dass nur Dateien mit der Erweiterung txt angezeigt werden.


4 Verändere die Eigenschaften des SaveDialog1 ebenso.

5 Definiere unter `var` im Kopf des Programms die Variable Nummer als globale Variable. Sie legt die Anzahl der Elemente der Arrays fest:

```
var
 Nummer: Integer;
```

6 Klicke doppelt auf den Button „Laden“. Füge den Quelltext in die Prozedur ein und ergänze ihn. Beachte die Kommentare!

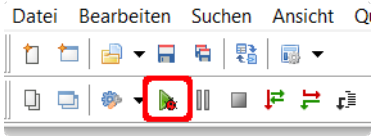
```
var
  Zaehler: Integer;

begin
   // den Variablen einen Startwert zuweisen
   Nummer:= 0;
   Zaehler:= 0;
   // Titel des Dialogs festlegen
   OpenDialog1.Title:= 'Datei öffnen ...';
   // OpenDialog1 aufrufen
   if OpenDialog1.Execute then
   begin
     // Memo/Listbox leeren
     Memo1.Clear;
     ListBox1.Clear;

     // die ausgewählte Datei (FileName) öffnen
     // zeilenweise lesen und in das Memo-Feld übertragen
     Memo1.Lines.LoadFromFile(Opendialog1.FileName);

     // Inhalt des Memo-Felds in das Array übertragen
     // solange lesen, bis letzte Zeile erreicht ist
     while Zaehler< Memo1.Lines.Count-1 do
     begin
        // Zeile jeweils einem Array zuordnen
        // 4 Zeilen bilden einen Block für den Datensatz
        // deshalb:
        // Vorname → aktueller Zaehler
        // Nachname → aktueller Zaehler + 1
        // Ort → aktueller Zaehler + 2
        // Strasse → aktueller Zaehler + 3
        Vorname[Nummer]:= Memo1.Lines[Zaehler];
        . . .

        // Nachname der Listbox hinzufügen
        ListBox1.Items.Add(Nachname[nummer]);

        // ein Datensatz umfasst 4 Zeilen
        // Datensatz gelesen -> 4 Zaehler vor
        // nächster Datensatz -> Nummer um 1 erhöhen
        Zaehler:= Zaehler + 4;
        inc(Nummer);
     end;
  end;
end;
```

7 Das Speichern ist etwas einfacher:
Zunächst werden alle Daten in das Memo-Feld geschrieben. Das ist deswegen notwendig, weil sich der Inhalt des Memo-Felds durch Anklicken eines ListBox-Eintrags verändert haben kann.

```
Memo1.Clear;
for Zaehler:= 0 to Nummer - 1 do
  begin
     Memo1.Lines.Add(Vorname[Zaehler]);
     Memo1.Lines.Add(Nachname[Zaehler]);
     . . .
  end;
```

Dann wird der Dialog zum Speichern aufgerufen und der Inhalt des Memo-Felds in die Datei geschrieben (SaveToFile).

```
// Titel des Dialogs
SaveDialog1.Title:= 'Datei speichern ...';
if SaveDialog1.Execute then
Memo1.Lines.SaveToFile(SaveDialog1.FileName);
```

8 

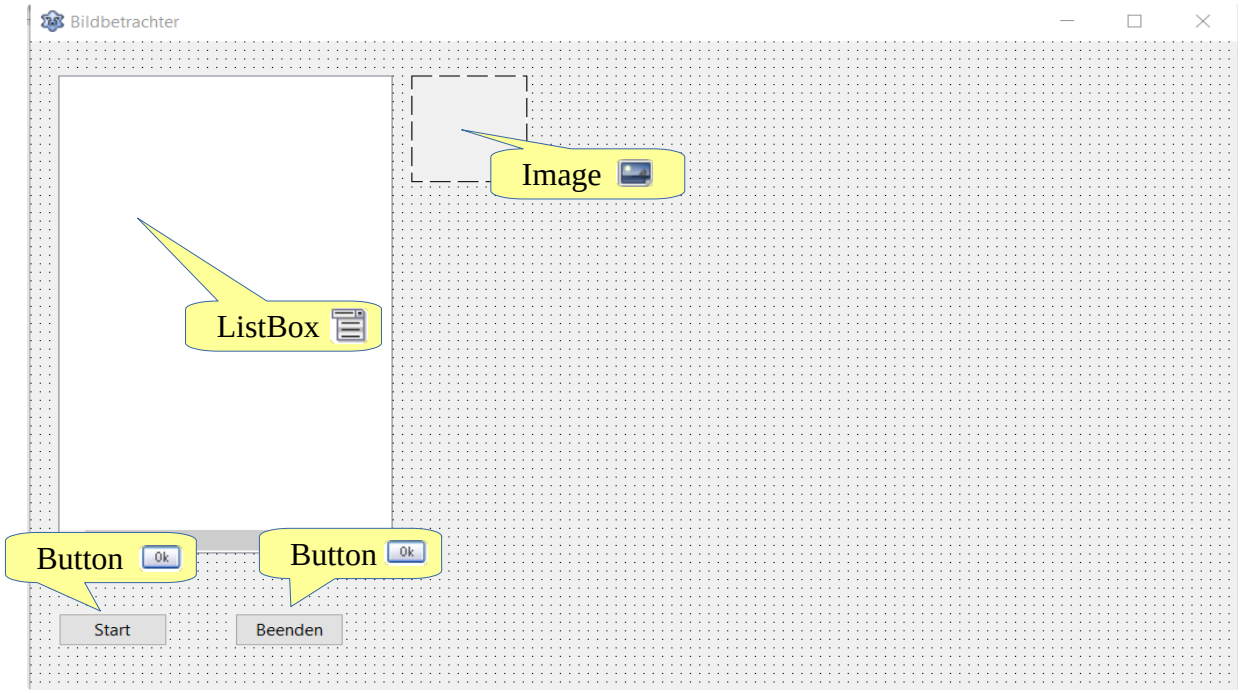
Fertig, du kannst das Programm starten.

# Dateien lesen

## Minimalistischer Bildbetrachter

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen. Du musst die Fotos und das Programm im gleichen Ordner speichern.

1 Erstelle die Form und beschrifte die Buttons.




Das Symbol für die Statuszeile findest du hier.

2 Erstelle ein dynamisches String-Array:

```
DateiName: Array of String;
```

3 Erstelle durch einen Doppelklick auf den Button „Start“ eine entsprechende Prozedur.

Für das Lesen der Bilddateien wird ein Record verwendet. Records ermöglichen es, mehrere Variablen zu gruppieren. Dies ist beispielsweise dann hilfreich, wenn oft die gleiche Menge an Variablen benötigt wird, oder eine Menge Variablen logisch zusammengefasst werden soll.

In diesem Fall wird der bereits definierte Record `TsearchRec` verwendet, er beinhaltet die Zeit (Time), die Größe (Size) und den Namen einer Datei (Name).
Für eine erfolgreiche Suche benötigst du noch FindFirst und FindNext.

```
var
  SuchErgebnis: TsearchRec;
  Zaehler: Integer;
begin
    Zaehler:= 0;
    // faAnyFile → beliebige Datei
    // *.jpg → Suchmaske
    // erstes (FindFirst) Element suchen, das der Suchmaske entspricht
    if FindFirst('*.jpg', faAnyFile, SuchErgebnis) = 0 then
    begin
       repeat
       begin
            // dynamisches Array um ein Element erhöhem
            SetLength(DateiName, Zaehler + 1);
            // dem Dateinamen des Suchergebnisses wird ein
            // Element des Arrays DateiName zugewiesen
            DateiName[Zaehler]:= SuchErgebnis.Name;
            // aktuellen Dateinamen der Listbox hinzufügen
            ListBox1.Items.Add(DateiName[Zaehler]);
            // Zaehler um 1 erhöhen (inc=increase)
            inc(Zaehler);
       end;

       // wiederholen, solange eine Datei gefunden wurde
       until FindNext(SuchErgebnis) <> 0;

       // Speicher freigeben
       FindClose(SuchErgebnis);
    end;
 end;
```

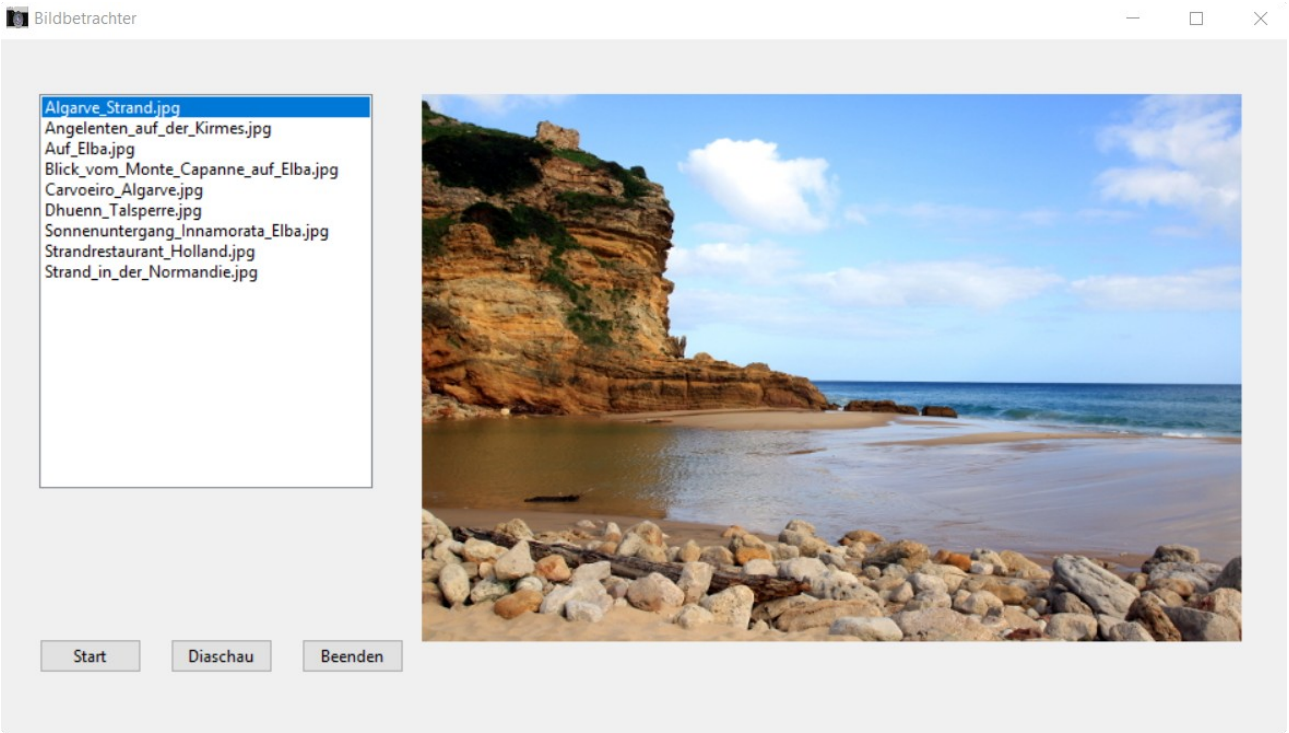
Beim Klick in die Listbox wird das entsprechende Bild geladen:

```
// Breite und Höhe des Bildes
Image1..Width:= 800;
Image1.Height:= 600;
// Proportionen des Bildes erhalten
Image1.Proportional:= true;
// erlauben, das Bild über die tatsächliche Größe hinaus zu vergrößern
Image1.Stretch:= true;
// aktuell markiertes Bild laden
Image1.Picture.LoadFromFile(ListBox1.Items.Strings[ListBox1.ItemIndex]);
```

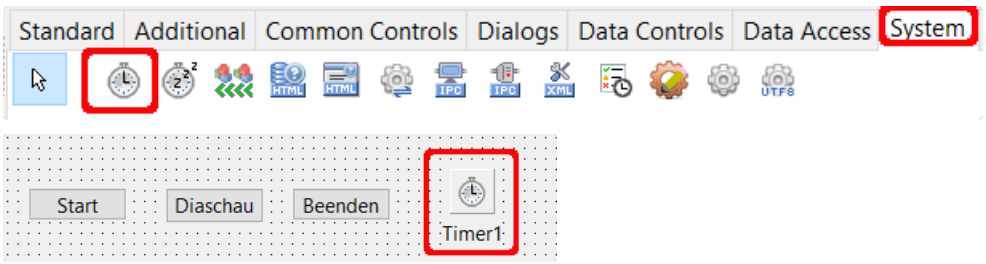
# Timer verwenden

## Diaschau

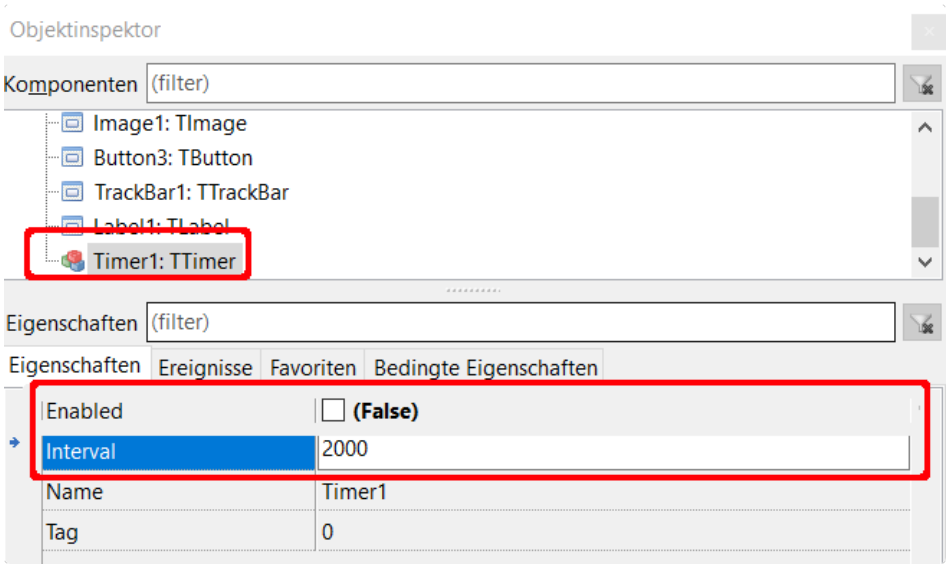
Die Fotos kannst du hier herunterladen. Du musst sie in deinem Projektordner speichern. Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.




1 Setze das Timer-Symbol in die Form. Es ist nur in der Entwurfsansicht sichtbar.



2 Setze die Eigenschaft `Interval` des Timers auf 2000 (entspricht 2000 Millisekunden → 2 Sekunden) und die Eigenschaft `Enabled` auf false.



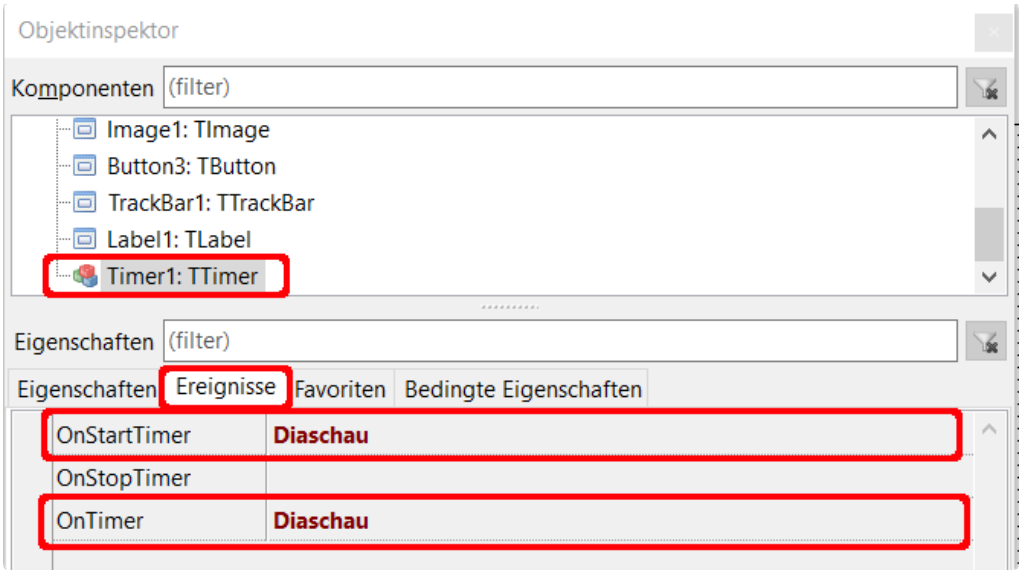

3 Nach einem Doppelklick auf den Button „Diaschau“ musst du in der Prozedur den Timer aktivieren:

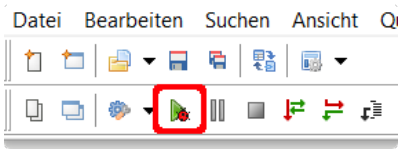
```
Timer1.Enabled:= true;
```

4 Erstelle eine neue Prozedur Diaschau. Beachte die Kommentare.

```
procedure TForm1.Diaschau(Sender: Tobject);
// IndexListe ist die aktuelle Position in der Listbox
var IndexListe: Integer;

begin
   // ListBox1.ItemIndex ist der gerade markierte Eintrag
   IndexListe:= ListBox1.ItemIndex;

   // Diaschau soll solange laufen, bis der letzte Eintrag der Liste
   // erreicht ist →  ListBox1.Count - 1
    if IndexListe < ListBox1.Count - 1 then
    begin
       // nach Erhöhen um 1 der IndexListe wird auch der markierte Eintrag
       // der Liste um 1 erhöht
       inc(IndexListe);
       ListBox1.ItemIndex:= IndexListe;
       // das Bild, das zum aktuelle markierten Eintrag gehört
       // ListBox1.Items.Strings[ListBox1.ItemIndex] wird geladen
       Image1.Picture.LoadFromFile(ListBox1.Items.Strings[ListBox1.ItemIndex]);
   end;
end;
```

5 Im letzten Schritt musst du noch dafür sorgen, dass die Diaschau auch gestartet wird, wenn der Timer aktiviert wird.




6 


Das Programm ist jetzt startbereit.

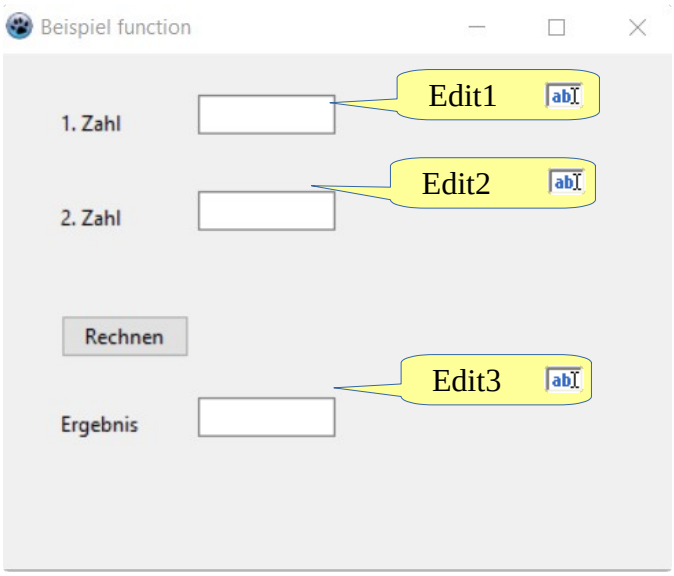
# Function – Befehlsabläufe wieder verwenden

Mit dem Aufruf einer Funktion kann eine Abfolge von Befehlen immer wieder ausgeführt werden.
**Beispiel**:

```
function ProduktZweiZahlen(zahl1, zahl2: Integer): Integer;
begin
   Result := Zahl1 * Zahl2;
end;
```

Im Kopf der Funktion wird der Name festgelegt, in den Klammern werden die Variablen und der Variablentyp notiert (zahl1 und zahl2 vom Type Integer), hinter der Klammer wird der Typ der Variable des Rückgabewertes definiert. Das Schlüsselwort Result gibt den ermittelten Wert (hier das Produkt zweier Zahlen) zurück.
Im Programm wird die Funktion aufgerufen:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Ergebnis: Integer;
begin
   // die Funktion erwartete Integer
   // → Textfelder müssen zu Integer umgewandelt werden (StrToInt)
   Ergebnis:= ProduktZweiZahlen(StrToInt(Edit1.Text), StrToInt(Edit2.Text));
   // Ergebnis muss wieder zu String umgewandelt werden (IntToStr)
   Edit3.Text:= IntToStr(Ergebnis);
end;
```

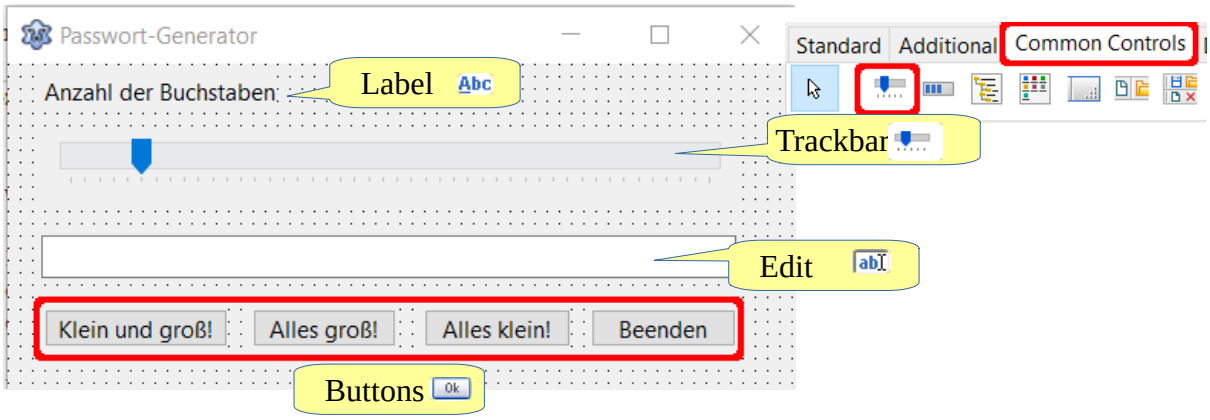



Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

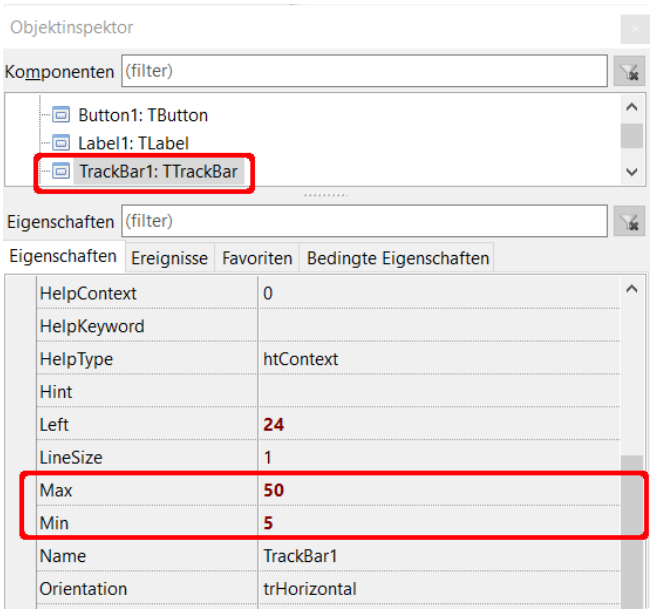
# Passwortgenerator

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Erstelle die Form und beschrifte die Buttons und das Label.




2 Ändere die Eigenschaften **Min** und **Max** der Trackbar1:




3 Erstelle eine Funktion `PasswortErzeugen`. Definiere die Variablen:

AlleZeichen → Zeichen, die im Passwortvorkommen dürfen
Passwort → das erzeugte Passwort
Zaehler → Schleifenvariable
Position → zufällige Position in den erlaubten Zeichen

```
function PasswortErzeugen(Laenge: Integer): String;
// Laenge → aktuelle Position der Trackbar1
// Rückgabewert → String
var
  AlleZeichen, Passwort: String;
  Position, Zaehler: Integer;
begin
   // erlaubte Zeichen
   AlleZeichen:= 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!?-_%&$';
   // Passwort leeren
   Passwort:= '';
   // Zufallsgenerator starten
   Randomize;
   // Passwort zusammensetzen
   // von 1 bis zur aktuellen Position der Trackbar
   for Zaehler:= 1 to Laenge do
   begin
      // zufällige Position im String AlleZeichen bestimmen
      // Length → Länge des Strings
      Position:= Random(Length(AlleZeichen)) + 1;
      // Passwort zusammensetzen:
      // bereits bestehendes Passwort + durch Zufall ermitteltes Zeichen
      // aus AlleZeichen
      // copy →  copy(String, Position_im_String, Länge)
      Passwort := Passwort + copy (AlleZeichen, Position, 1);
   end;
   Result:= Passwort;
end;
```

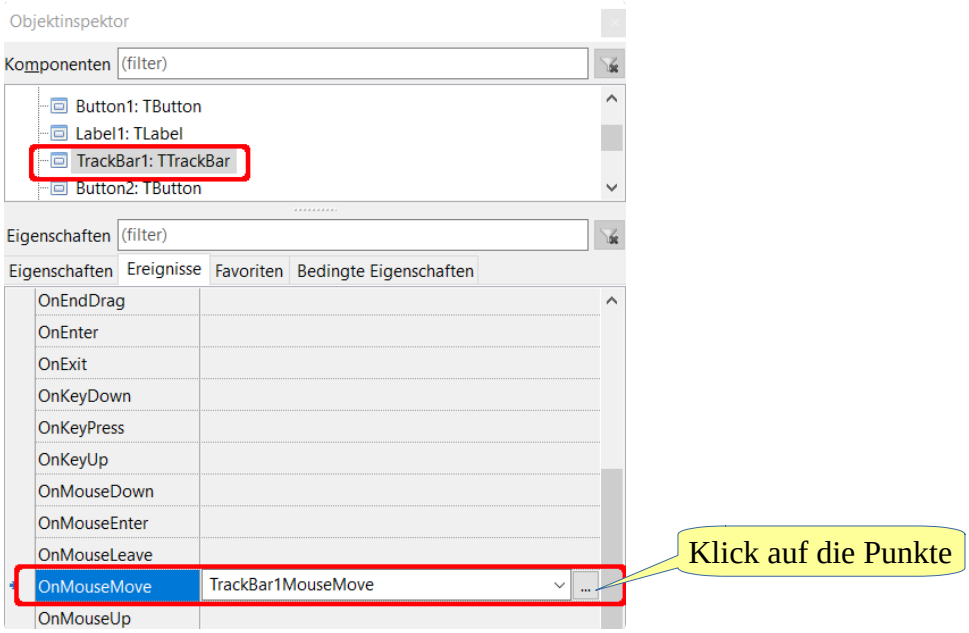
4 Ein Doppelklick auf die Buttons erzeugt die zugehörigen Klick-Prozeduren. Hier wird immer die Prozedur PasswortErzeugen aufgerufen, als Länge des Passwortes wird die Position der Trackbar übergeben:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   Edit1.Text:= PasswortErzeugen(Trackbar1.Position);
end;
```

Für die Großbuchstaben wird der Befehl `UpperCase`, für die Kleinbuchstaben `LowerCase` vorangestellt.

```
Edit1.Text:= UpperCase(PasswortErzeugen(Trackbar1.Position));
Edit1.Text:= LowerCase(PasswortErzeugen(Trackbar1.Position));
```

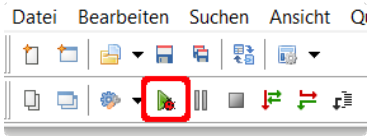
5 Jetzt fehlt noch die Beschriftung des Labels mit der aktuellen Position der Trackbar:




In der Prozedur wird dem Label eine Beschriftung zugewiesen:

```
Label2.Caption:= IntToStr(Trackbar1.Position);
```

6

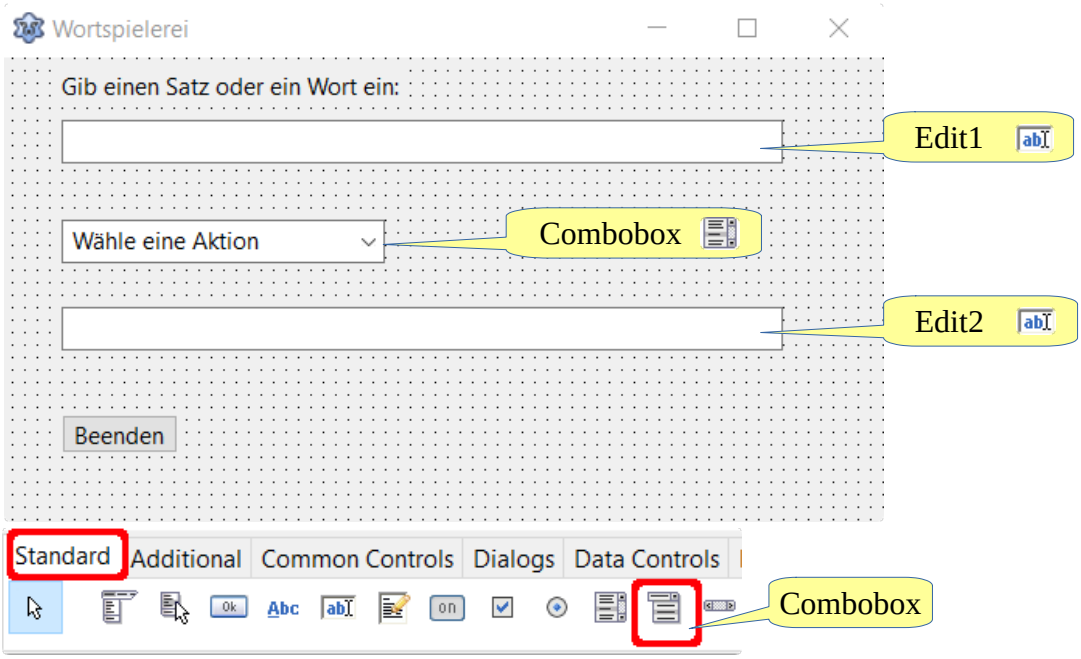



Du kannst das Programm starten.

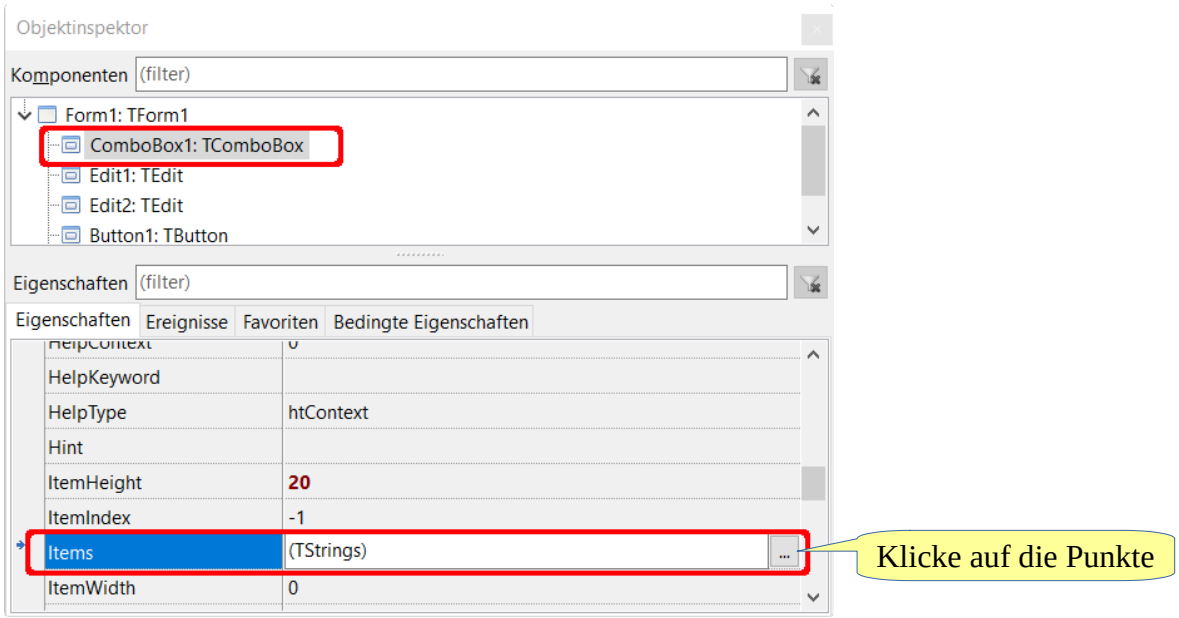
# Spielereien mit Buchstaben und Worten

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

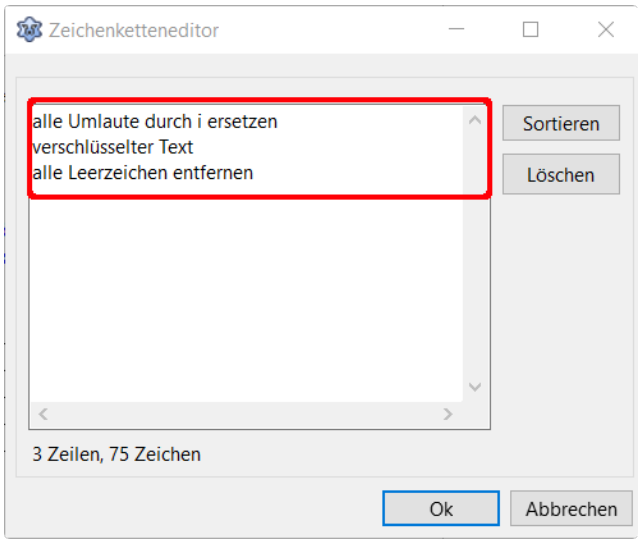
1 Erstelle die Form.




2 Ändere die Eigenschaften der Combobox:



3




Bearbeite mit dem Zeichenketteneditor den Auswahlmöglichkeiten der Combobox.

4

Die Funktion BuchstabeErsetzen wird mit mehreren Parametern aufgerufen:

| | |
|---|---|
| OriginalText | → der eingegebene Text aus Edit1 |
| ZuErsetzenderText | → der Buchstabe, der ausgetauscht werden soll |
| ErsatzText | → der Buchstabe, der den ausgetauschten Buchstaben ersetzt |

Hinter der Klammer wird der Typ des Rückgabewertes festgelegt (String)
Der Befehl Pos durchsucht einen String nach einem Zeichen und gibt dann die Position dieses Zeichens zurück. Wird es nicht gefunden, wird 0 zurück gegeben.

```
function BuchstabeErsetzen(OriginalText, ZuErsetzenderText, ErsatzText: String):
String;
var
Position, Zaehler: Integer;
begin
   // jedes Zeichen des Textes untersuchen, Length → Länge des Textes
   for Zaehler:= 0 to Length(OriginalText) do
   begin
      // Position des zu ersetzenden Zeichens bestimmen
      Position:= Pos(ZuErsetzenderText, OriginalText);

      // nur wenn der gesuchte Buchstabe gefunden wurde
      if Position > 0 then
      begin
        // Zeichen löschen mit delete
        // an der aktuellen Position mit der Länge des Ersatzbuchstaben
        Delete(OriginalText, Position, Length(ZuErsetzenderText));

        // neues Zeichen einfügen mit insert
        // der Ersatzbuchstabe wird an der aktuellen Position eingefügt
        Insert(ErsatzText, OriginalText, Position);
      end;
      // Rückgabe des geänderten Textes mit Result
      Result:= OriginalText;
   end;
end;
```

Diese Funktion wird bei jeder Auswahl in der Combobox aufgerufen.

5 Der ausgewählte Eintrag in der Combobox wird mit case abgefragt:

```
case ComboBox1.ItemIndex of
// alle Umlaute austauschen
0:   . . .
// die Buchstaben a e i o u werden jeweils durch ihre Position im Alphabet
//    // ausgetauscht
// a → 1, e → 5, i → 9, o → 15, u → 21
1:   . . .
// Leerzeichen entfernen
2: . . .
```

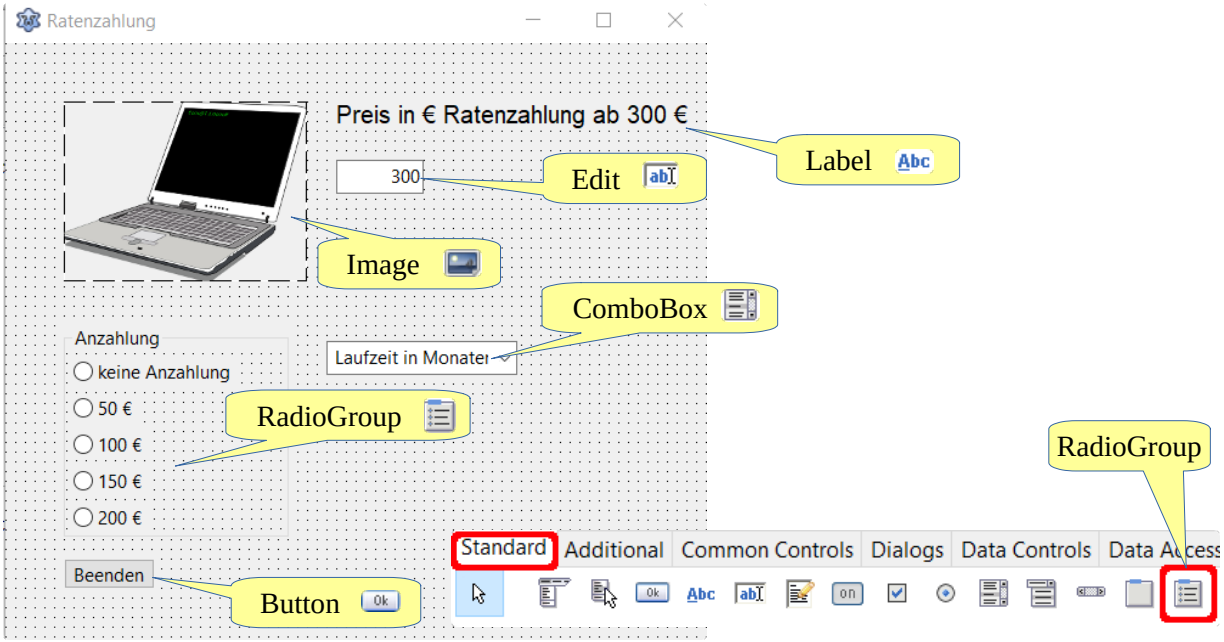
Der Aufruf der Funktionen unterscheidet sich lediglich durch die Angabe der Buchstaben, die ausgetauscht werden sollen und der Angabe des Ersatzbuchstaben. Sowohl Austausch- als auch Ersatzbuchstaben dürfen auch mehrere Buchstaben sein. Beim ersten Aufruf der Funktion wird der Inhalt des Textfeldes Edit1 verwendet, bei den folgenden wird der zuvor geänderte Text modifiziert.

```
OriginalText:= BuchstabeErsetzen(Edit1.Text, 'a', 'i');
OriginalText:= BuchstabeErsetzen(OriginalText, 'e', 'i');
. . .
```

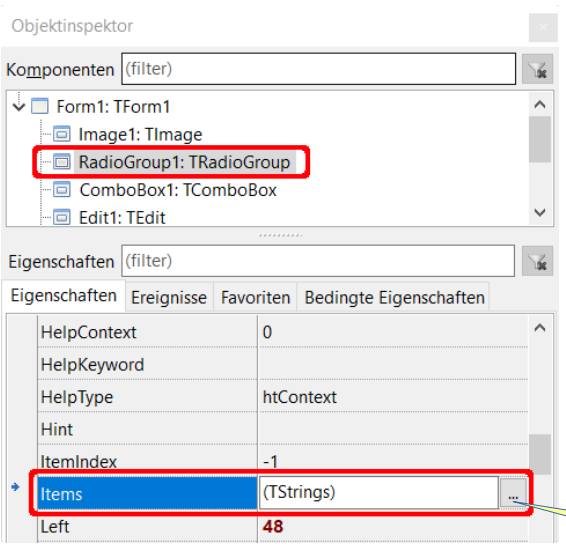
## Ratenzahlungen berechnen

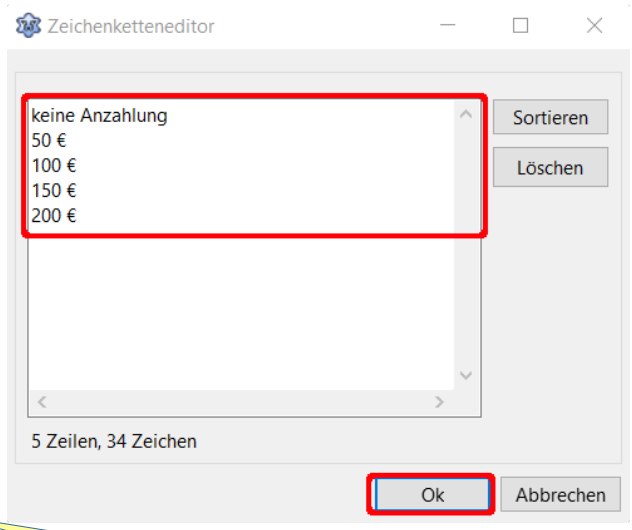
Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Erstelle die Form. Das Bild findest du hier.

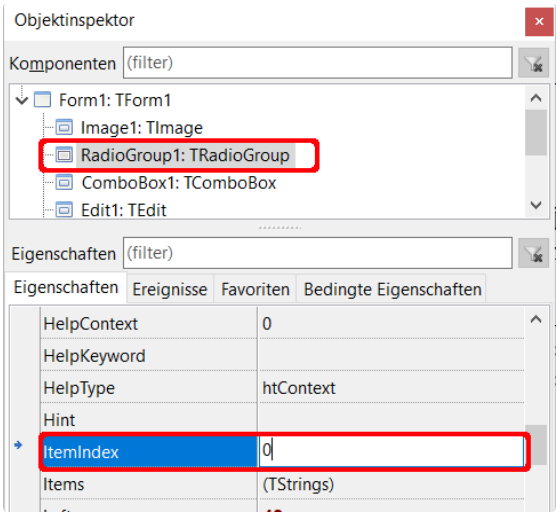


2 Die Einträge für die Auswahlmöglichkeiten der RadioGroup finden sich im Objektinspektor → Eigenschaften → Items



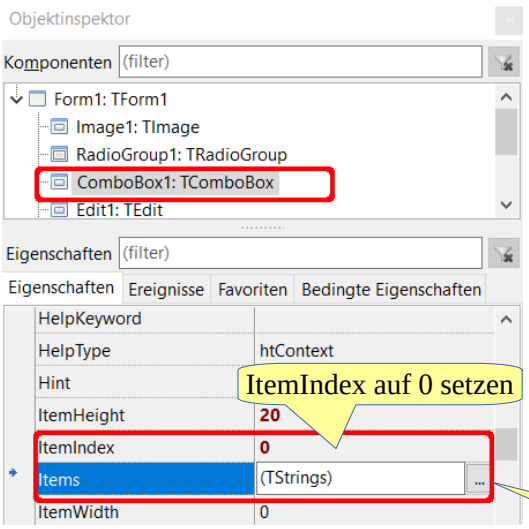




Klicke auf die Punkte

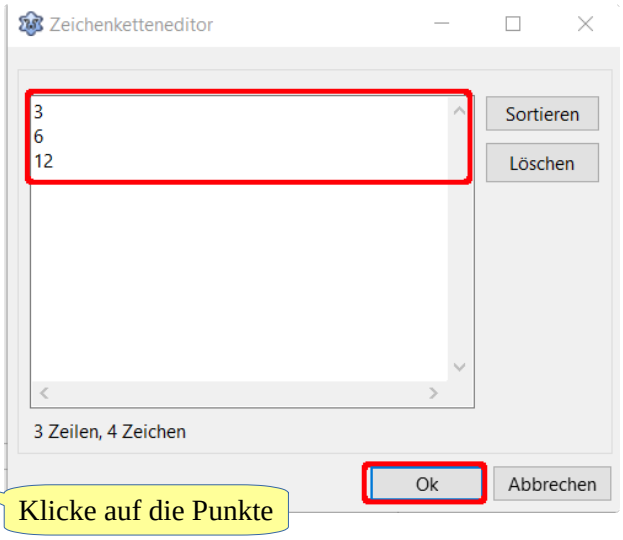



Setze den ItemIndex auf 0, dann wird der erste Radiobutton markiert.

3




ItemIndex auf 0 setzen




Klicke auf die Punkte

4 Die Rate soll aus der Anzahlung, der Laufzeit und dem Gesamtpreis berechnet werden. Weil das Ergebnis der Berechnung eine Dezimalzahl sein kann, werden alle Variable als Double definiert. Außerdem müssen sie im gesamten Programm gültig sein.

```
var
  Anzahlung: Double;
  Laufzeit, Rate, GesamtPreis: Double;
```

5 Ein Doppelklick in die RadioGroup erstellt die dazugehörige Click-Prozedur.
Mit `case` wird dem jeweiligen Radiobutton ein Wert zugewiesen:
Die Zählung beginnt mit 0.

```
case Radiogroup1.ItemIndex of
   0: Anzahlung:= 0;
   1: Anzahlung:= 50;
   . . .
   . . .
end;
```

6 Die Funktion RateBerechnen gibt als Ergebnis die Höhe der Rate zurück. Als Variable werden die Anzahlung, die Laufzeit und der Gesamtpreis vom Typ Double übergeben. Der Rückgabewert ist eine Variable vom Typ Double.

```
function RateBerechnen(Anzahlung, Laufzeit, GesamtPreis: Double): Double;
begin
   Result:= (GesamtPreis - Anzahlung) / Laufzeit;
end;
```

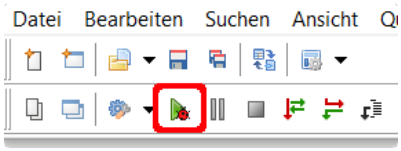
7 Ein Doppelklick auf die ComboBox erstellt die Prozedur ComboBox1Change. Die Rate wird durch Aufruf der Funktion RateBerechnen berechnet:

```
// Preis aus dem Edit-Feld lesen
GesamtPreis:= StrToFloat(Edit1.Text);
// Combobox1.Items.Strings[ComboBox1.ItemIndex] → aktuelle Wert der
// ComboBox → muss zu Double umgewandelt werden
// Die Variable Rate muss noch definiert werden
Rate:= RateBerechnen(Anzahlung,
StrToFloat(Combobox1.Items.Strings[ComboBox1.ItemIndex]), GesamtPreis);
```

Sorge dafür, dass nach zwei Nachkommastellen gerundet wird. Du musst vorher die Bibliothek `Math` unter `uses` einbinden:

```
Rate:= RoundTo(Rate, -2);
```

8 Jetzt musst du noch im Label die Höhe der Rate ausgeben.
Auch beim Wechsel der Radiobuttons soll die Rate berechnet werden. Füge unterhalb der case-Abfrage die Berechnung der Rate ein.

9 

Das Programm ist jetzt startbereit.

Ein Problem gibt es noch: Wenn der Kaufpreis kleiner als 300 € ist, macht eine Ratenzahlung keinen Sinn. Das kannst du verhindern, indem du vor der Berechnung der Rate eine if-Abfrage setzt:

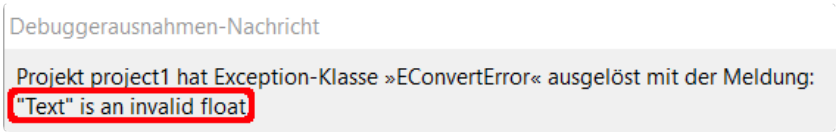
```
// Gesamtpreis muss größer/gleich 300 sein
if (Gesamtpreis >=300) then
begin
   // Rate berechnen
   . . .
end;
```

Übrigens lässt sich die Rate auch in einem Schritt berechnen:

```
Label2.Caption:= 'Monatliche Rate: ' + FloatToStr(RoundTo(RateBerechnen(Anzahlung,
StrToFloat(Combobox1.Items.Strings[ComboBox1.ItemIndex]), GesamtPreis),-2)) + ' €';
```

# try .. except: Fehler abfangen

Ein typischer Fehler ist der Versuch der Umwandlung eines Strings, der Buchstaben enthält, in eine Zahl mit StrToFloat. Das Programm zeigt zur Laufzeit einen Fehler an.




Diesen Fehler kann man mit try … except „abfangen“.
Im try-Block stehen die abzuarbeitenden Anweisungen. Im except-Block stehen die Anweisungen, wie im Fehlerfall auf den Fehler zu reagieren ist.

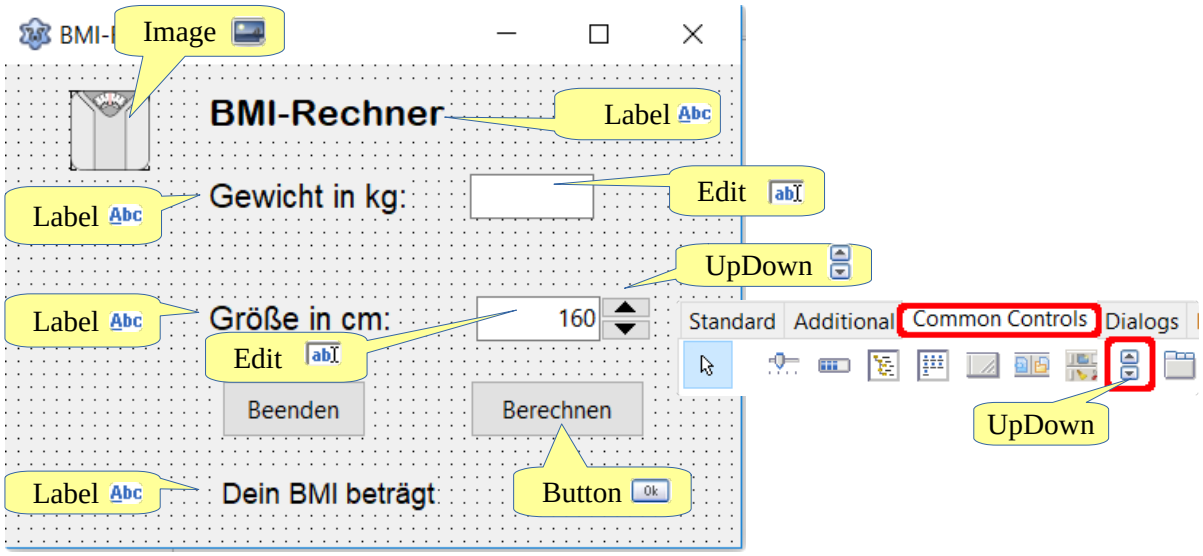
Aufbau einer try … except Anweisung:

```
try
   // zu prüfende Anweisungen
   . . .
   // Fehlerbehandlung
except
   . . .
end;
```

# BMI-Rechner

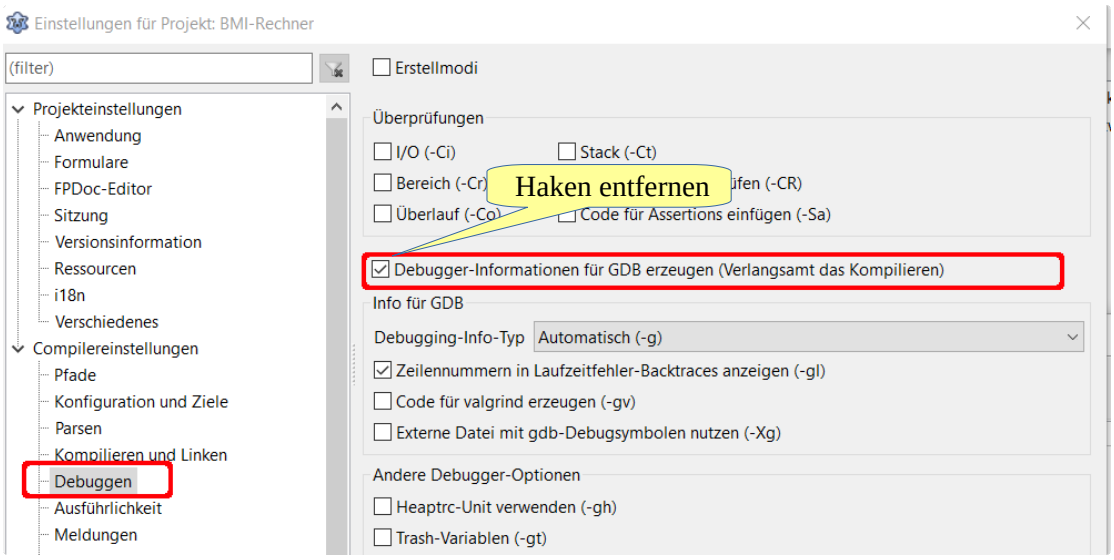
Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen. Das Bild findest du hier.

1 Erstelle die Form.

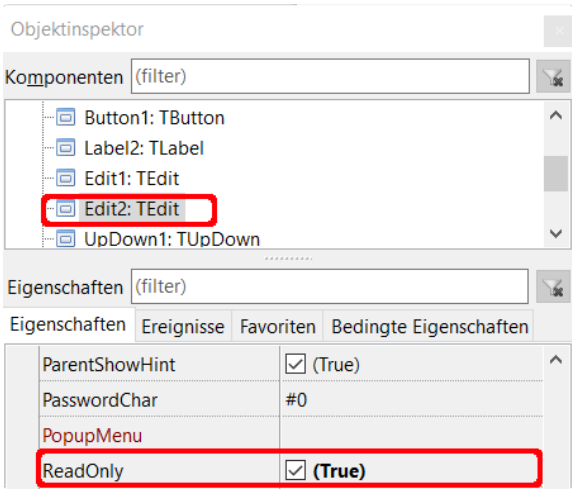



2 Lazarus bringt einen sogenannten „Debugger“ mit, der bei Fehlern im Programm eine Meldung ausgibt. DieAnweisung, die unter `try` stehen. werden nicht ausgeführt, stattdessen wird der `except`-Teil ausgeführt und die Fehlermeldung angezeigt.

Zum Testen musst du für dieses Programm unter Projekt → Projekteinstellungen den Debugger ausschalten:

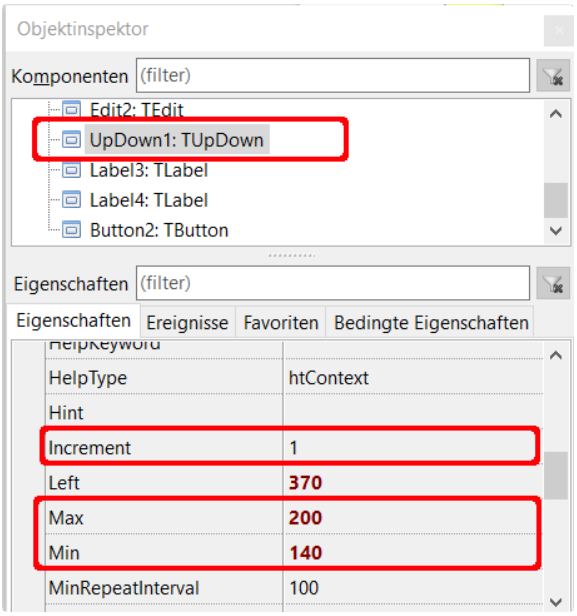


3 Setze die Eigenschaften des Textfeldes Edit2 Readonly auf True, damit die Größe nur über das UpDown-Feld verändert werden kann.

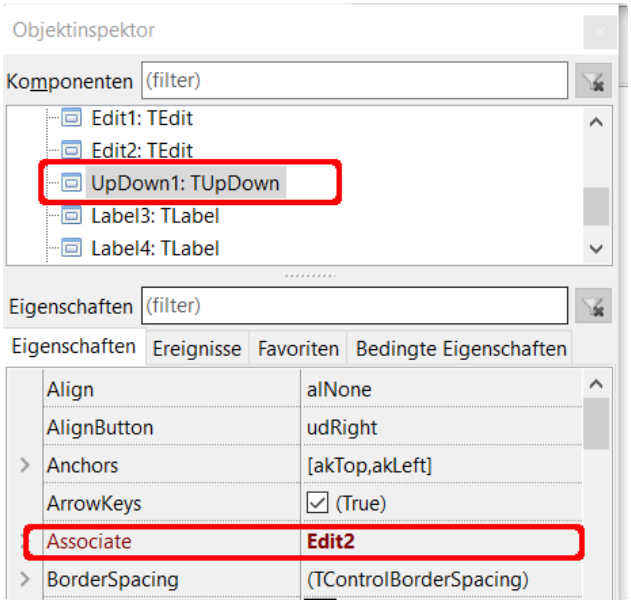



Verändere noch bei beiden Edit-Feldern die Eigenschaft Alignment auf taRightJustify.


4 UpDown benötigt noch die Schrittweite (Increment), den kleinst- und den größtmöglichen Wert. (Min/Max):



5 Der Wert des UpDown soll im Textfeld Edit2 angezeigt werden. Die einfachste Möglichkeit ist die Verknüpfung (Associate):



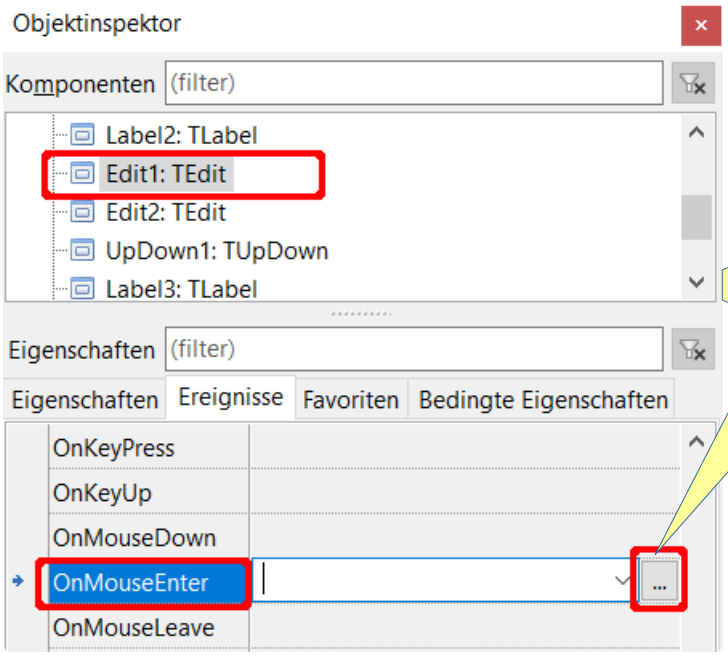

Alternativ kannst du auch mit einem Doppelklick auf UpDown1 die Click-Prozedur erzeugen:

```
procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
   Edit2.Text:= IntToStr(UpDown1.Position);
end;
```

6 Erstelle mit einem Doppelklick die Click-Prozedur zum Button berechnen.
**Beachte die Kommentare.**

```
procedure TForm1.Button1Click(Sender: TObject);
var
BMI: Double;
begin
   // versuche, den BMI zu berechnen
   // Körpergewicht in Kilogramm geteilt durch das Quadrat der Körpergröße
   // in Metern
   // für RoundTo -> unter uses Math hinzufügen
   // 0 → keine Nachkommastellen
   try
       BMI:= RoundTo(StrToFloat(Edit1.Text) / (StrToFloat(Edit2.Text)/100 *
       StrToFloat(Edit2.Text)/100), 0);
      Label2.Caption:= 'Dein BMI beträgt: ' + FloatToStr(BMI);
      // Berechnung ist nicht möglich, weil das Textfeld leer ist oder
      // falsche Zeichen enthält
   Except
     ShowMessage('Du musst das Gewicht eingeben!');
   end;
end;
```

7 Es wäre schön, wenn das Textfeld für die Eingabe des Gewichts automatisch geleert und der zuletzt berechnete BMI gelöscht würde, wenn die Maus dort hineinfährt. Das ist einfach zu bewerkstelligen:




Die Prozedur enthält zwei Befehle:

```
Label2.Caption:= '';
Edit1.Text:= '';
```

8 Du kannst das Programm jetzt starten.

## Taschenrechner

`try .. except` kann auch auf Fehler mit einer Fehlermeldung reagieren, die den Fehler genau beschreibt.

**Beispiele:**

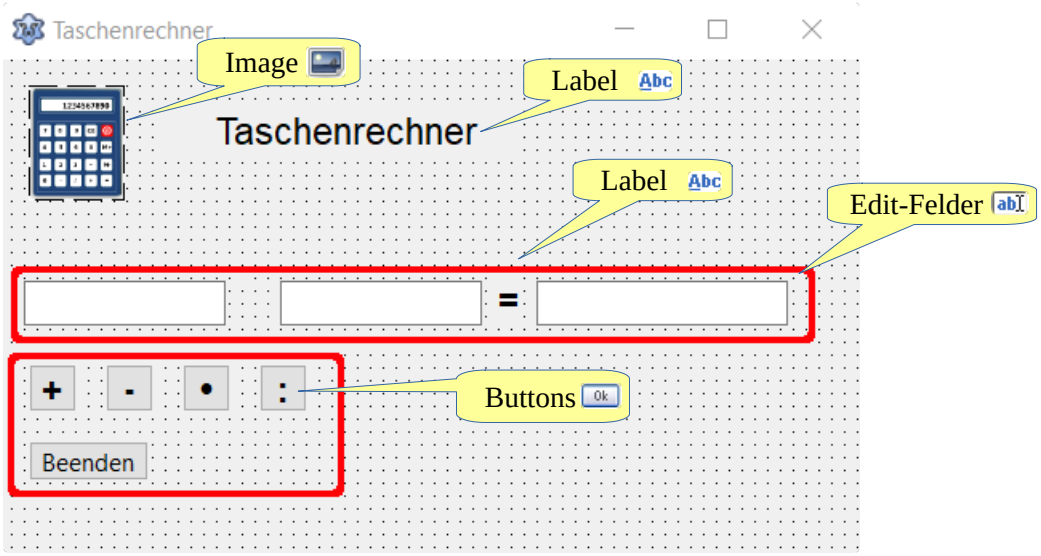
**EDivByZero**: Division durch 0

**EConvertError**: Fehler bei der Umwandlung von Variablentypen (StrToFloat/StrToInt). Die Umwandlung ist nicht möglich, wenn der String neben Zahlen auch andere Zeichen enthält.

**EInValidOp**: Ungültige Rechenoperation. Aus negativen Zahlen kann keine Wurzel gezogen werden.

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Erstelle die Form. Das Bild findest du hier.




2 Gib den Buttons aussagekräftige Namen; Plus, Minus, Mal und Geteilt. Erstelle mit einem Doppelklick auf den Plus-Button die Click-Prozedur.

```
procedure TForm1.PlusClick(Sender: TObject);
begin
   Edit3.Text:= '';
   // versuche die nächsten Befehle auszuführen
   try
      Zahl1:= StrToFloat(EingabeZahl1.Text);
      Zahl2:= StrToFloat(EingabeZahl2.Text);
      // wenn die Edit-Felder neben Zahlen auch andere Zeichen enthalten
      // weiter mit Except (EConvertError)
      Ergebnis:=Zahl1 + Zahl2;
      Edit3.Text:= FloatToStr(Ergebnis);
   Except
   // Fehler bei der Umwandlung
   on EConvertError do
   begin
      // Ergebnis-Feld leeren, Fehlermeldung ausgeben
      // ShowMessage öffnet ein Mitteilungsfenster
      Edit3.Text:= '';
      ShowMessage('Du darfst natürlich nur Zahlen eingeben!');
   end;
 end;
end;
```

2 Erstelle in gleicher Weise die Click-Prozeduren für Minus und Mal.

3 Bei der Division gibt es zusätzlich eine Besonderheit: Eine Division durch 0 ist nicht möglich. Du musst diesen Fehler mit einem weiteren `try .. except`-Block abfangen.

```
try
   Zahl1:= StrToFloat(EingabeZahl1.Text);
   Zahl2:= StrToFloat(EingabeZahl2.Text);
   // wenn die Edit-Felder neben Zahlen auch andere Zeichen enthalten
   // weiter mit Except (EConvertError)
Except
on EConvertError do
  ShowMessage('Du darfst natürlich nur Zahlen eingeben!');
end;

try
   Ergebnis:=Zahl1 / Zahl2;
   // bei Division durch 0
   // weiter mit Except (EDivByZero)
   Edit3.Text:= FloatToStr(Ergebnis);
Except
on EDivByZero do
   begin
     Edit3.Text:= '';
     ShowMessage('Division durch 0 ist nicht möglich!');
   end;
end;
```

4 Die Behandlung des Fehlers EDivByZero lässt sich beim Starten des Programms im Entwurfsmodus nicht beobachten. Du musst in deinem Projektordner die entsprechende exe-Datei starten.

5 Fertig, das Programm ist startbereit.


Linux verwendet nicht das Komma, sondern stattdessen den Punkt als Dezimaltrennzeichen. Damit es nicht zu einer Fehlermeldung kommt, muss bei jeder Rechenoperation das Komma durch einen Punkt ersetzt werden.

```
procedure TForm1.PlusClick(Sender: TObject);
begin
  . . .
  try
    EingabeZahl1.Text:= StringReplace(EingabeZahl1.Text, ',', '.', [rfReplaceAll, rfIgnoreCase]);
    EingabeZahl2.Text:= StringReplace(EingabeZahl2.Text, ',', '.', [rfReplaceAll, rfIgnoreCase]);
    Zahl1:= StrToFloat(EingabeZahl1.Text);
    Zahl2:= StrToFloat(EingabeZahl2.Text);
    . . .
```

# Records – Variable zusammenfassen

Ein Record ist ein Datensatz, der aus verschiedenen Elementen besteht. Die Elemente können auch aus unterschiedlichen Datentypen bestehen.

Beispiel:
```
// Beginn der Deklaration
// Bezeichnung des Datensatzes
type
Namen = Record
   Vorname: String;
   PLZ: Integer;
   Ort: String;
   Strasse: Integer;
   Hausnummer: Integer;
end;
```

Unter var wird eine Variable dem Record zugewiesen:
```
var
  Person: Namen;
```

Zugriff auf die Elemente des Records;

```
Person.Vorname:=  'Klaus';
Person.Nachname:= 'Weber';
Person.PLZ:= 12345;
Person.Ort:= 'Kiel';
Person.Strasse:= 'Hauptstr.';
Person.Hausnummer:= 12;
```

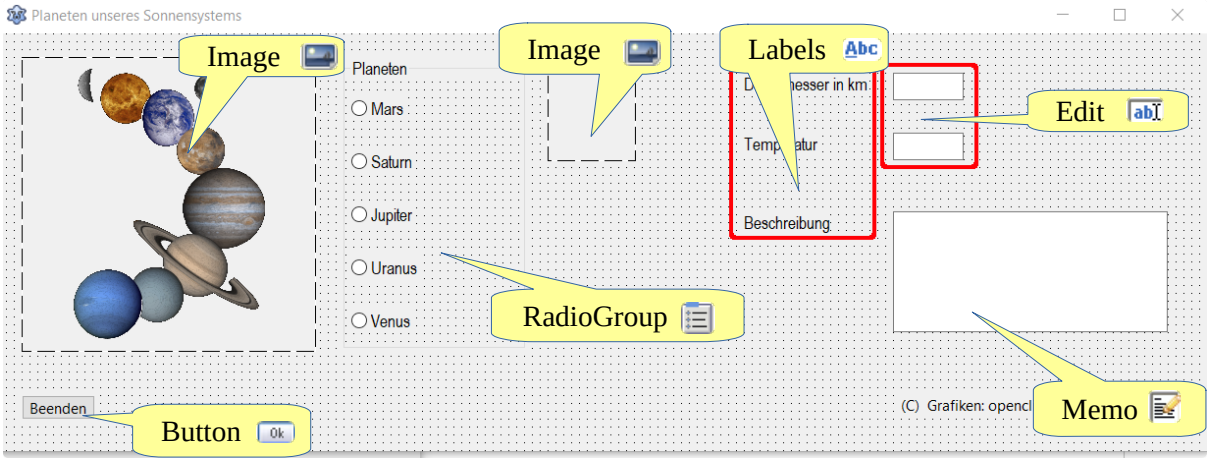
Natürlich können auch Arrays verwendet werden:

```
Namen: Array [0..4] of Person;

// erster Datensatz
Person[0].Vorname:=  'Laura';
Person[0].Nachname:= 'Schmitz';
Person[0].PLZ:= 98765;
Person[0].Ort:= 'München';
Person[0].Strasse:= 'Isarstr.';
Person[0].Hausnummer:= 32;

// zweiter Datensatz
Person[1].Vorname:=  'Klaus';
Person[1].Nachname:= 'Weber';
Person[1].PLZ:= 12345;
Person[1].Ort:= 'Kiel';
Person[1].Strasse:= 'Hauptstr.';
Person[1].Hausnummer:= 12;
. . .
. . .
```

# Unser Sonnensystem

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen. Wenn du das Programm ausprobieren willst, musst du die Bilder (mars.png, saturn.png, venus.png und jupiter.png) im gleichen Ordner wie die exe-Datei speichern.

1 Erstelle die Form. Das Bild findest du hier.




2 Definiere einen Record und unter var ein Array:

```
type
   // Name des Records
   Himmelskoerper = record
   // Elemente
   Durchmesser: Integer;
   Temperatur: Integer;
   Beschreibung: String;
   Bild: String;
 end;

var
Planet: Array [0..4] of Himmelskoerper;
```

3 Jetzt brauchst du die Daten. Du kannst sie hier herunterladen und in eine Prozedur DatenLaden packen..

```
procedure DatenLaden;
begin
  . . .
end;
```

Du musst den Namen der Prozedur selbst in die Liste der Prozeduren im Kopf des Programms eintragen!

4 Die Prozedur DatenLaden wird am Ende des Quelltextes aufgerufen:

```
begin
  DatenLaden;
end.
```

5 Erstelle mit einem Doppelklick auf die RadioGroup die dazugehörige Click-Prozedur.

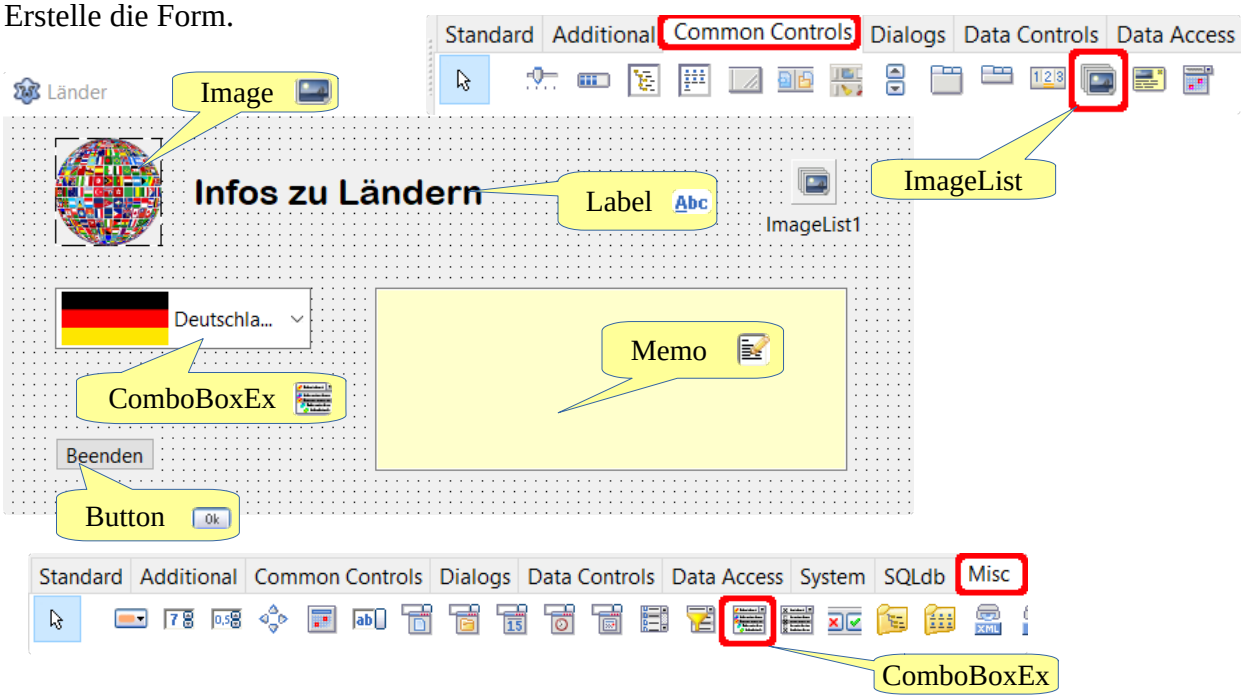
```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
   // Text zum Label Durchmesser
   Edit1.Text:= IntToStr(Planet[Radiogroup1.ItemIndex].Durchmesser);
   // Text zum Label Temperatur
   Edit2.Text:= IntToStr(Planet[Radiogroup1.ItemIndex].Temperatur);
   Memo1.Text:= Planet[Radiogroup1.ItemIndex].Beschreibung;
   // Bild anzeigen
   Image2.Picture.LoadFromFile(Planet[Radiogroup1.ItemIndex].Bild);
end;
```

6 Das Programm ist jetzt startbereit.

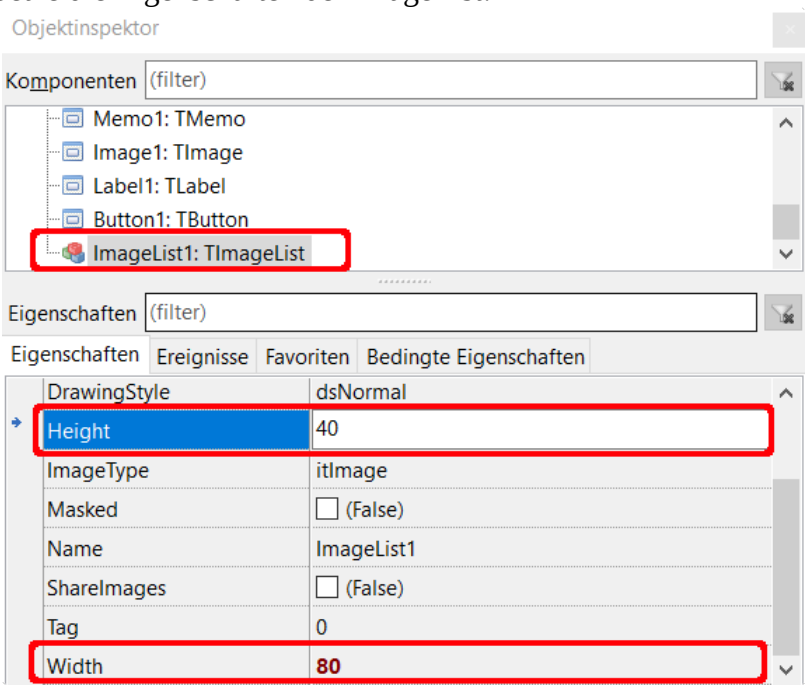
# Informationen zu Ländern anzeigen

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

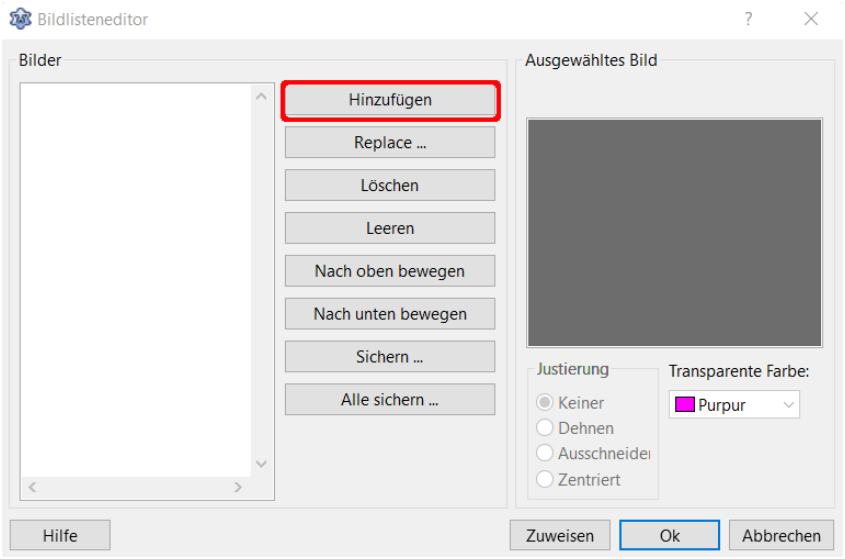
1 Erstelle die Form.



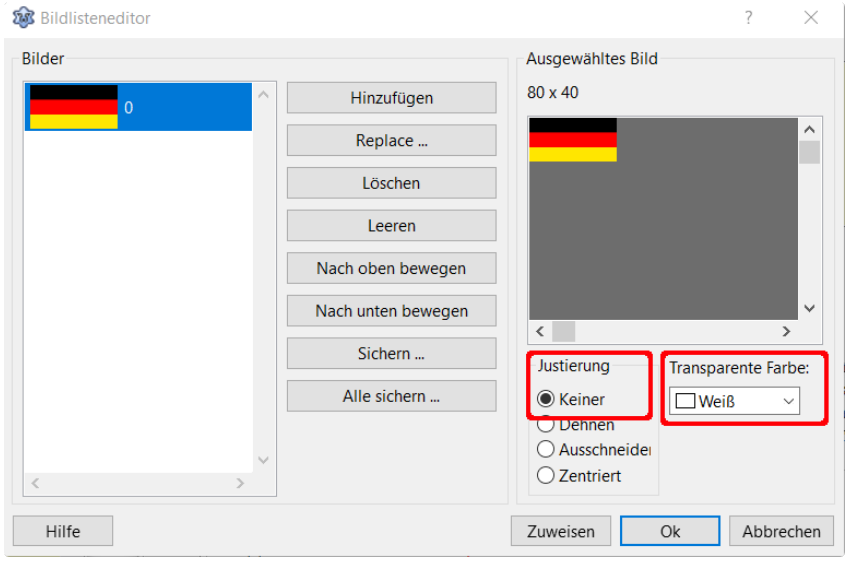

2 Setze die Eigenschaften der ImageList:



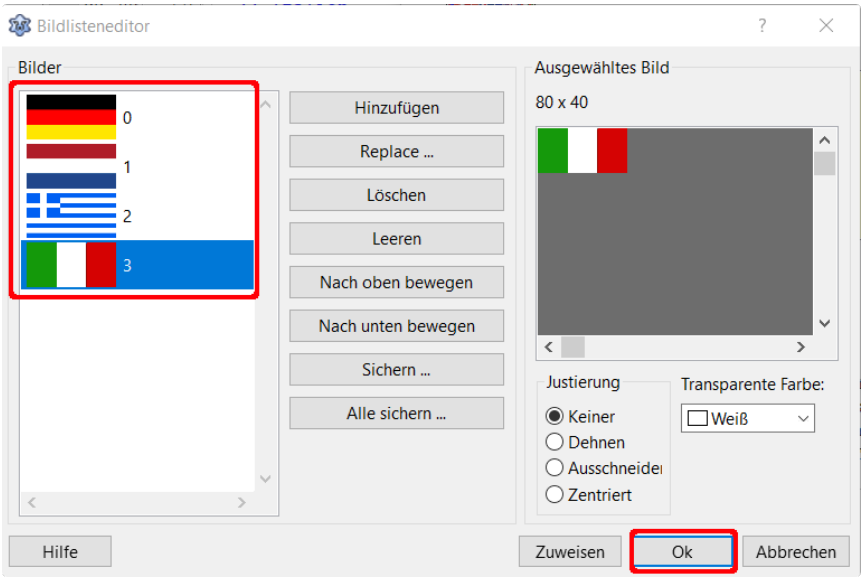
3 Lade die Bilder deutschland.jpg, niederlande.jpg, griechenland.jpg und italien.jpg herunter und speichere sie in deinem Projektordner.
Doppelklick auf den Button ImageList1. Füge nacheinander die Bilder ein:



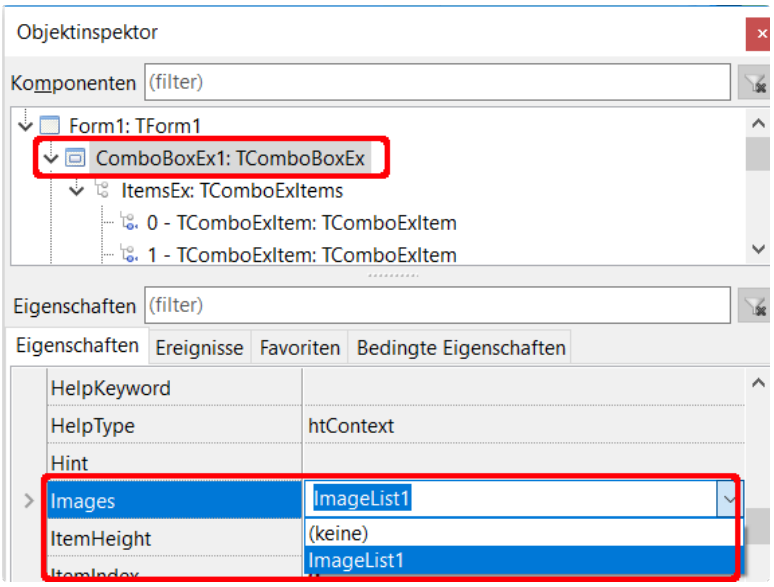

Achte darauf, dass bei jedem Bild die Transparente Farbe auf weiß gestellt ist:



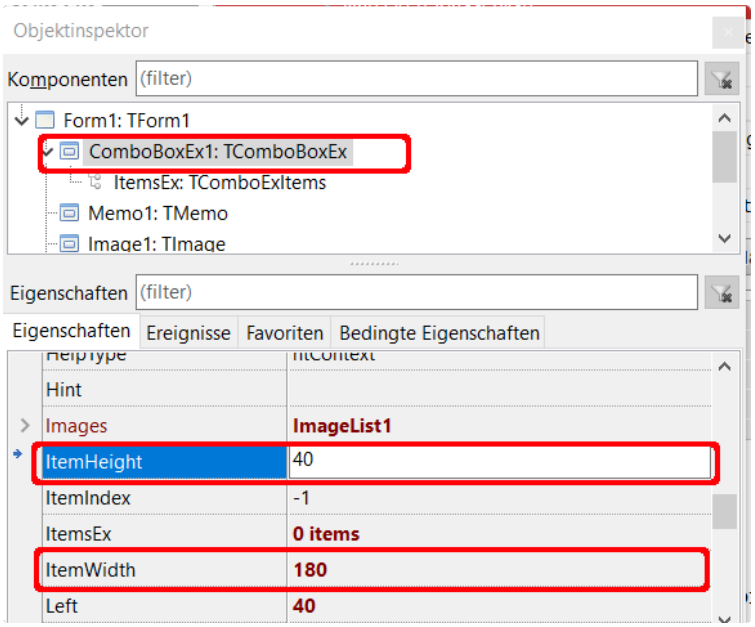
So muss es am Ende aussehen:



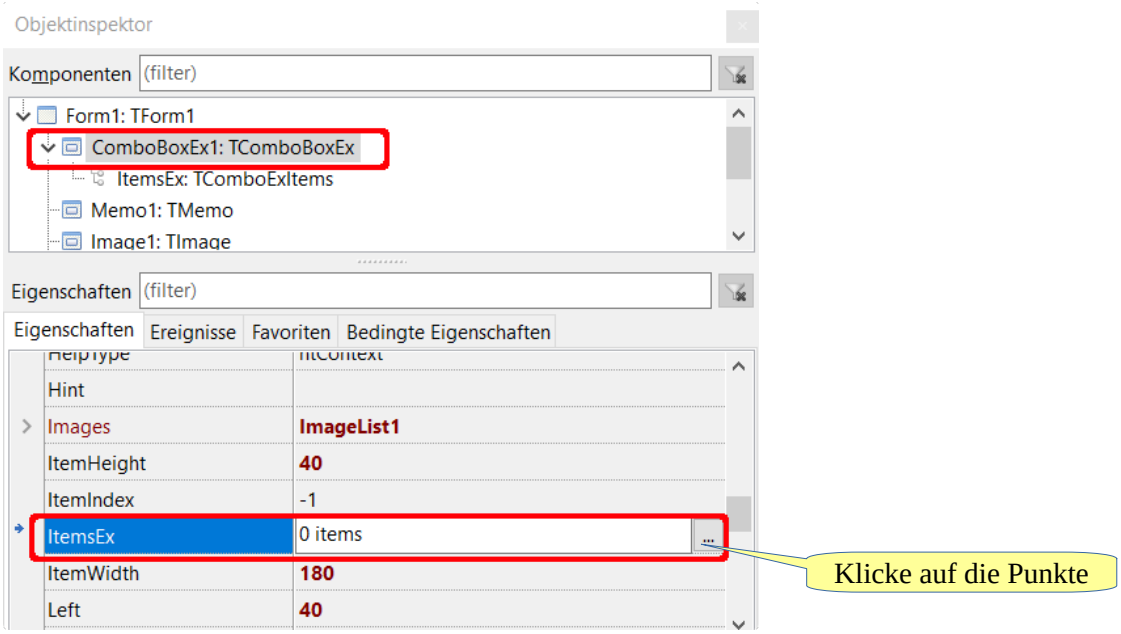

4 Setze die Eigenschaft Images auf ImageList1:



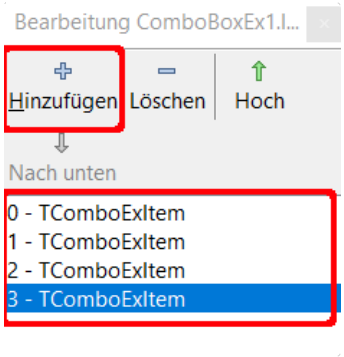
5 Lege die Höhe und Breite der ComboBox1 fest:



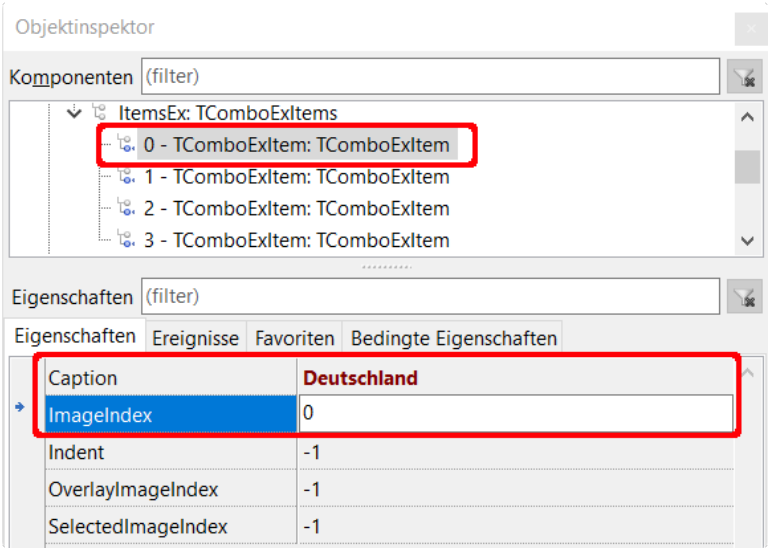

6 Erstelle die Einträge für die ComboBox1:



Erstelle vier Einträge:

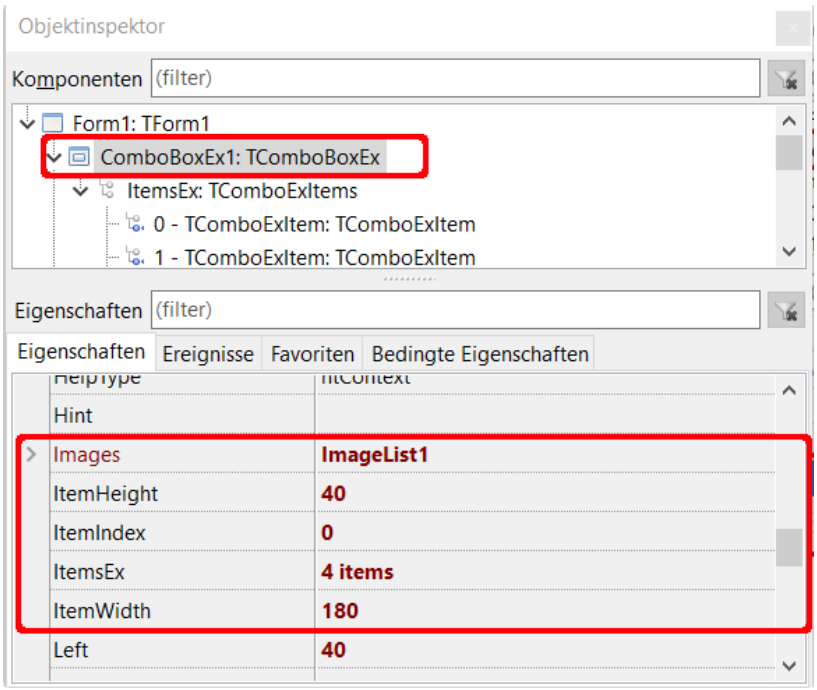



Jedem Eintrag muss jeweils ein Text (Caption) und ein Index (die Nummer des Bildes) zugewiesen werden:




0 – TComboExItem → Deutschland → 0
1 – TComboExItem → Niederlande → 1
2 – TComboExItem → Griechenland → 2
3 – TComboExItem → Italien → 3

Die Eigenschaften der ComboBoxEx1 sehen am Ende so aus:
Zusätzlich wurde hier die Eigenschaft ItemIndex auf 0 gesetzt, damit der erste Eintrag sofort angezeigt wird.




6 Das Bild findest du hier.

7 Jetzt kann die Programmierung beginnen. Definiere zunächst unter den Zeilen uses den Record Laender ...

```
type
  Laender = record
  Flaeche: Integer;
  Hauptstadt: String;
  Einwohner: Double;
  BevoelkerungsDichte : Integer;
end;
```

und anschließend das Array unter var.

```
var
Land: Array [0..4] of Laender;
```

Du kannst auch gleich unter uses die Bibliothek Math einbinden. Sie wird später für das Runden von Zahlen benötigt.

8 Die Daten werden wieder in einer Prozedur DatenLaden abgelegt:

```
procedure DatenLaden
begin
   // Deutschland
   Land[0].Flaeche:= 357385;
   Land[0].Hauptstadt:= 'Berlin';
   Land[0].Einwohner:= 82175684;
   Land[0].BevoelkerungsDichte := 230;
   // Niederlande
   Land[1].Flaeche:= 41548;
   Land[1].Hauptstadt:= 'Amsterdam';
   Land[1].Einwohner:= 16979120;
   Land[1].BevoelkerungsDichte := 408;
   // Griechenland
   Land[2].Flaeche:= 131957;
   Land[2].Hauptstadt:= 'Athen';
   Land[2].Einwohner:= 10995000;
   Land[2].BevoelkerungsDichte := 83;
   // Italien
   Land[3].Flaeche:= 301338;
   Land[3].Hauptstadt:= 'Rom';
   Land[3].Einwohner:= 60599000;
   Land[3].BevoelkerungsDichte := 201;
end;
```

9 Ein Doppelklick auf die ComboBoxEx erstellt die Click-Prozedur:

```
procedure TForm1.ComboBoxEx1Change(Sender: TObject);
var
   Zaehler: Integer;
begin
   Memo1.Clear;
   // Zaehler -> aktuelle gewählter Eintrag der Liste
   Zaehler:= ComboBoxEx1.ItemIndex;

   // Memo schreiben
   // für RoundTo -> Math unter uses hinzufügen
   Memo1.Lines.Add('Größe in km²:  ' + IntToStr(Land[Zaehler].Flaeche));
   Memo1.Lines.Add('Hauptstadt:  ' + Land[Zaehler].Hauptstadt);

   // Einwohnerzahl im Format xx,xx Millionen → / 1000000
   // Einwohner → Double → FloatToStr
   Memo1.Lines.Add('Einwohnerzahl in Millionen: ' +
   FloatToStr(RoundTo(Land[Zaehler].Einwohner/1000000, -2)));

   // Bevoelkerungsdichte → Integer → IntToStr
   Memo1.Lines.Add('Bevölkerungsdichte pro km²:  ' +
   IntToStr(Land[Zaehler].BevoelkerungsDichte));
end;
```

10 Als letzten Schritt müssen jetzt die unter DatenLaden abgelegten Daten auch gelesen werden.
Füge unmittelbar vor dem letzten end. (die letzte Zeile des Programms) den Aufruf der Prozedur ein:

```
begin
   DatenLaden;
```

11 Geschafft, das Programm ist startbereit.

Wende dein Wissen am folgenden Beispiel an.
Die Bilder findest du hier. (Die Autospezialisten mögen mir meine Fehler verzeihen)

| Für die ComboBoxEx: | Großes Bild |
|---|---|
| Fiat_500_klein.png | Fiat_500.png |
| VW_Golf_klein.png | VW_Golf.png |
| Mercedes_klein.png | Mercedes_png |
| VW_Scirocco_klein.png | VW_Scirocco.png |
| BMW_318_klein.png | BMW_318.png |

# Boolean – wahr oder falsch

Neben den Variablen String, Integer und Double, die du bisher kennengelernt hast, gibt es noch den Typ Boolean. Boolean kann nur zwei Werte annehmen: wahr (true) oder falsch (false). Viele Eigenschaften aus dem Objektinspektor kennen diese Werte:

```
Label1.Visible := true;
Memo1.Visible := false;
Image1.AutoSize:= true;
```

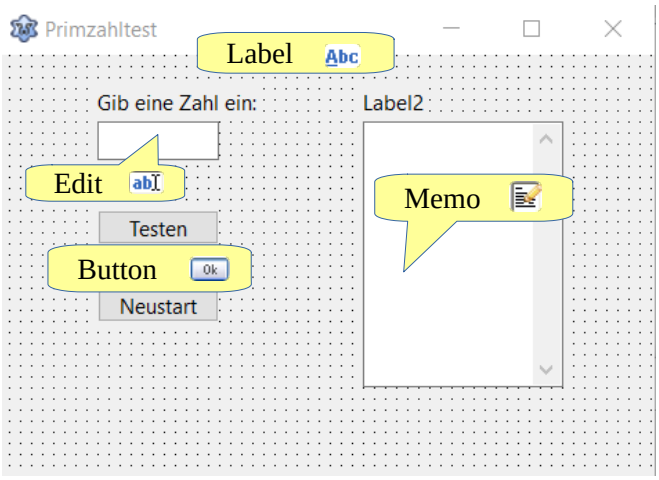
Natürlich kannst du auch selbst Variable vom Type Boolean definieren und ihnen einen Wert zuweisen:

```
var
  PrimZahl: Boolean

PrimZahl := true;
```

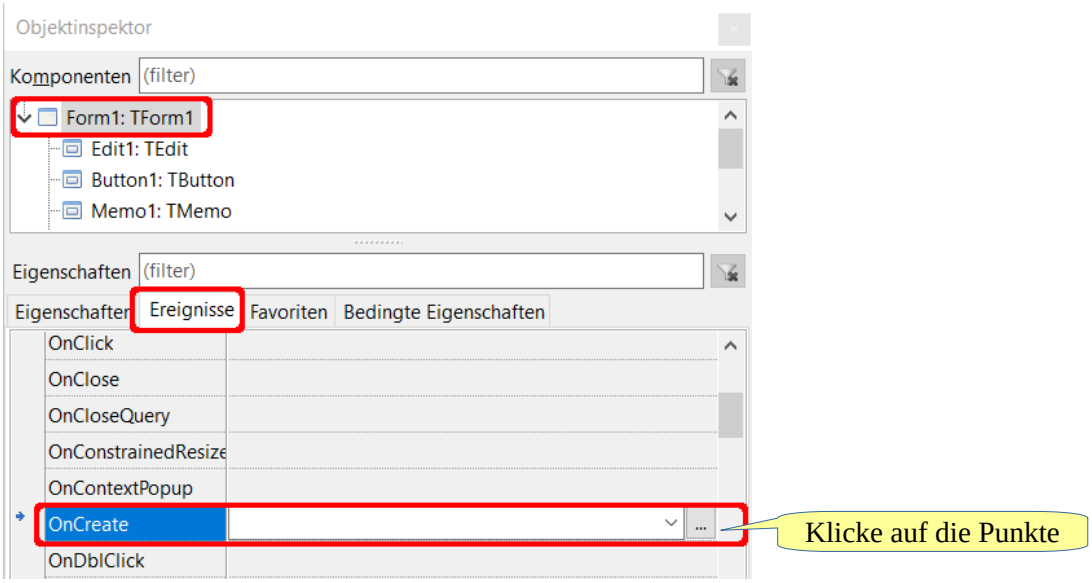
# Primzahltester

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Erstelle die Form und beschrifte die Buttons. Label2 wird später durch Leeren der Caption nur zur Laufzeit des Programms sichtbar




Lösche die Eigenschaft Caption von Label2, sie soll leer sein. Außerdem musst du die Eigenschaft ReadOnly von Memo1 auf true setzen.




Du kannst diese Schritte auch durch den Aufruf einer Prozedur erledigen:
Die Prozedur `FormCreate` wird beim Start des Programms aufgerufen:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   Memo1.ReadOnly:= true;
   Label1.Caption:= '';
end;
```

2 Erstelle eine Funktion PrimzahlTest.
Beachte die Kommentare.

```
// Zahl aus Edit1 wird als Integer übergeben
// Rückgabewert → String
function PrimzahlTest(Zahl: Integer): String;
var
   GroessterTeiler, Zaehler: Integer;
begin
   // der größtmögliche Teiler ist die Hälfte der zu prüfenden Zahl
   // div → Division von Integer → liefert als Ergebnis eine ganze Zahl
   GroessterTeiler:= Zahl div 2;
   AnzeigeZahl:= '';

   for Zaehler:= 2 to GroessterTeiler do
   begin
      // ist Zahl ohne Rest (mod) durch Zaehler teilbar
      // wenn ja -> Teiler zu AnzeigeZahl hinzufügen,
      // anschließend neue Zeile(sLineBreak)
      if Zahl mod Zaehler = 0 then
      begin
        AnzeigeZahl:= AnzeigeZahl + IntTostr(Zaehler) + sLineBreak;
      end;
   end;
   // AnzeigeZahl als String zurückgeben
   Result:= AnzeigeZahl;
end;
```

3 Erstelle mit einem Doppelklick auf den Button „Testen“ die Prozedur Button1Click.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   // Zahl PrimZahl → Boolean → true
   // keine Primzahl → Boolean → false
   PrimZahl: Boolean;

   begin
   // Memo-Feld löschen
   Memo1.Clear;

   // Aufruf der Funktion PrimzahlTest
   AnzeigeZahl:= PrimzahlTest(StrToInt(Edit1.Text));
   If AnzeigeZahl = '' then PrimZahl:= true else PrimZahl:= false;

   // PrimZahl → true → Anzeige Zahl ist Primzahl
   // PrimZahl → false → Anzeige der Teiler im Memo-Feld
   If PrimZahl = false then
   begin
      Memo1.Lines.add(AnzeigeZahl + sLineBreak) ;
      Label2.Caption:= 'Teiler von ' + Edit1.Text + ' sind:'
   end
   else Label2.Caption:= Edit1.Text + ' ist eine Primzahl!';
end;
```

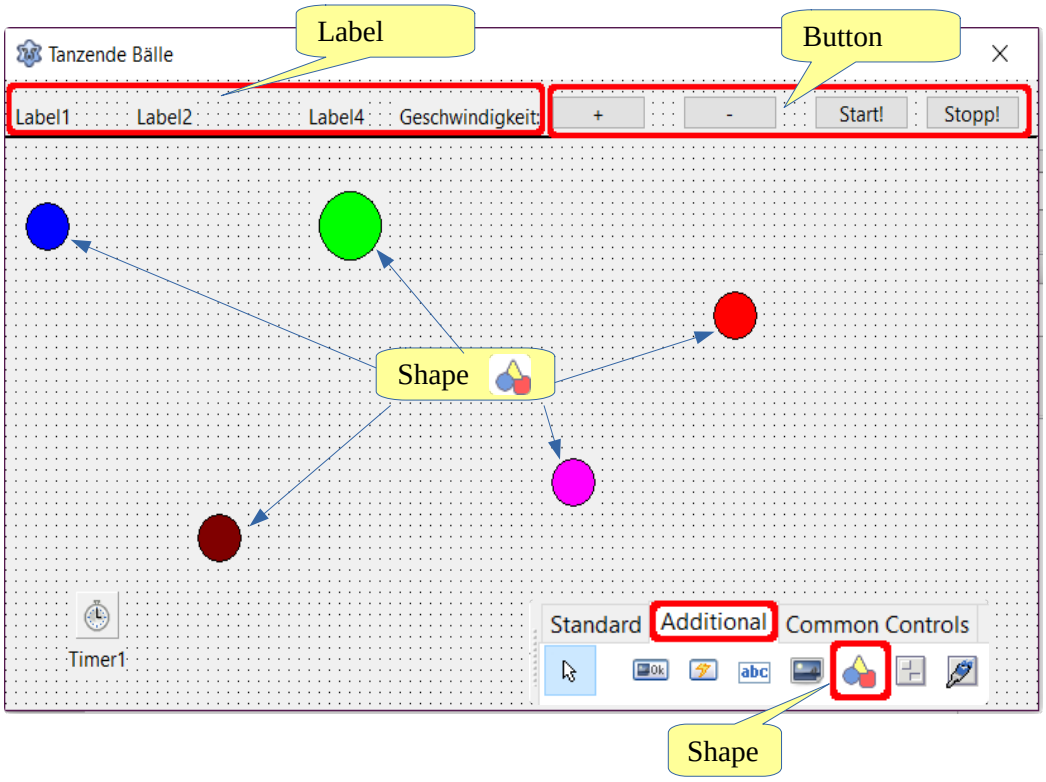
4 Fertig, du kannst das Programm starten.

## Tanzende Bälle

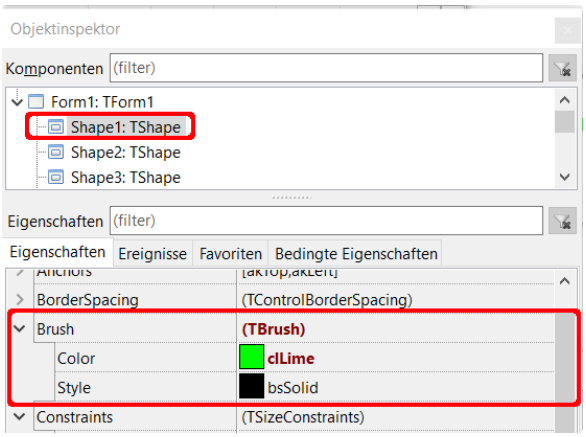
In diesem kleinen Spiel geht es darum, den grünen Ball anzuklicken.

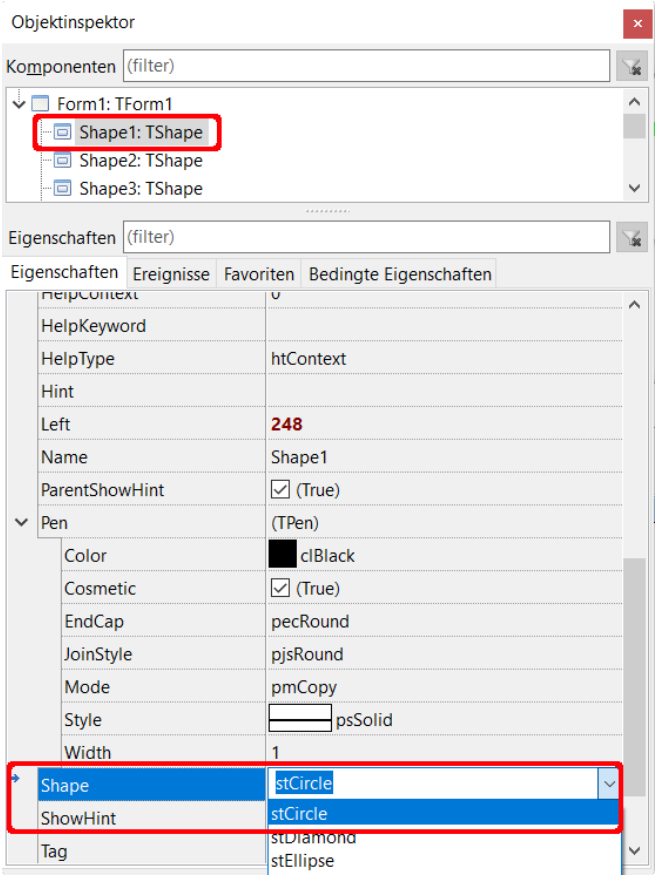
Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Erstelle die Form.



2 Im Objektinspektor änderst du die Form und Farbe der einzelnen Shapes:

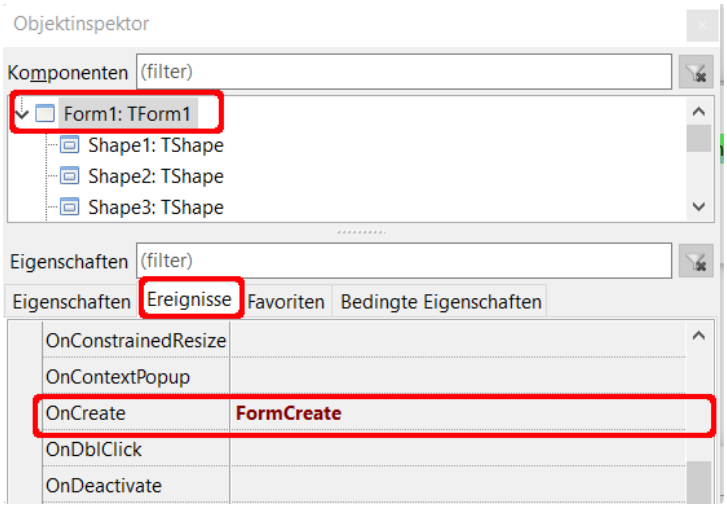



3 Definiere unter `var` die Variablen:

```
// Array der Shapes
Ball: array [1..5] of TShape;

// rechter Rand erreicht
Rechts: array [1..5] of Boolean;

// oberer Rand erreicht
Oben: array [1..5] of Boolean;
Zaehler: integer;

// Anzahl der Treffer
Treffer: Integer;

// Geschwindigkeit der Bälle
Geschwindigkeit: Integer;

// Zeitmessung
StartZeit: Double;
```

4 Füge unter den Ereignissen der Form die Prozedur `FormCreate` hinzu:




```
procedure TForm1.FormCreate(Sender: TObject);
begin
   // keine Treffer beim Start
   Treffer:= 0;
   // Array der Bälle
   Ball[1] := Shape1;
   Ball[2] := Shape2;
   Ball[3] := Shape3;
   Ball[4] := Shape4;
   Ball[5] := Shape5;
   // Start-Geschwindigkeit
   Geschwindigkeit:= 1;
   Label1.caption:= 'Treffer: ' + IntToStr(Treffer);
   Label2.Caption:= 'Geschwindigkeit: ' + IntToStr(Geschwindigkeit);
   // Zeitmessung starten
   StartZeit := GetTickCount64;
end;
```

5 Erstelle die Prozedur Start:

```
procedure TForm1.Start(Sender: TObject);
begin
   // abgelaufene Zeit anzeigen
   Label4.Caption:= FloatToStr(RoundTo((GetTickCount64 - StartZeit) /
   1000, -1)) + ' s';
   for Zaehler := 1 to 5 do
   begin
      // Rechts: = true → umdrehen nach rechts
      // Oben:= true → umdrehen nach oben
      // immer bezogen auf den aktuellen Ball Ball[Zaehler]

      // Untergrenze der Form erreicht → umdrehen
      if Oben[Zaehler] = True then
         Ball[Zaehler].Top := Ball[Zaehler].Top – Geschwindigkeit
      else
         Ball[Zaehler].Top := Ball[Zaehler].Top + Geschwindigkeit;

      // linke Kante der Form erreicht → umdrehen nach rechts
      if Rechts[Zaehler] = True then
         Ball[Zaehler].Left := Ball[Zaehler].Left + Geschwindigkeit
      else
         Ball[Zaehler].Left := Ball[Zaehler].Left - Geschwindigkeit;

      // Ball hat obere Begrenzung nicht erreicht
      if Ball[Zaehler].Top <= 40 then
         Oben[Zaehler] := False;

      // Ball erreicht untere Grenze der Form (Form1.Height)
      // obere Kante (top) + Höhe des Balls (Height)
      if Ball[Zaehler].Top + Ball[Zaehler].Height >= Form1.Height then
         Oben[Zaehler] := True;

      // linke Kante des Balls (Left) ist <= 0 -> linke Grenze der Form
      if Ball[Zaehler].Left <= 0 then
         Rechts[Zaehler] := True;

      // linke Kante des Balls (Left) + Breite des Balls (Width)
      // >= Breite der Form (Form1.Width)
      if Ball[Zaehler].Left + Ball[Zaehler].Width >= Form1.Width then
         Rechts[Zaehler] := False;
   end;
end;
```

Da du die Prozedur selbst erstellt hast, wurde sie nicht automatisch „angemeldet“. Du musst das selbst unter `type` erledigen:

```
procedure Start(Sender: TObject);
```

6 Jetzt fehlen noch die Prozeduren zu den Buttons „Start“, „Stopp“ und die Änderung der Geschwindigkeit.

Geschwindigkeit erhöhen
```
procedure TForm1.Button1Click(Sender: TObject);
begin
   inc(Geschwindigkeit);
   Label2.Caption:= 'Geschwindigkeit: ' + IntToStr(Geschwindigkeit);
end;
```

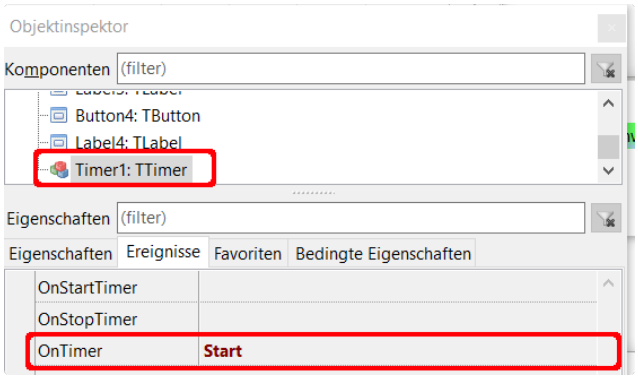
Geschwindigkeit verringern
```
procedure TForm1.Button2Click(Sender: TObject);
begin
   if Geschwindigkeit > 1 then dec(Geschwindigkeit);
   Label2.Caption:= 'Geschwindigkeit: ' + IntToStr(Geschwindigkeit);
end;
```

Programm anhalten
```
procedure TForm1.Button3Click(Sender: TObject);
begin
   Timer1.Enabled:= false;
end;
```

Programm starten
```
procedure TForm1.Button4Click(Sender: TObject);
begin
   Timer1.Enabled:= true;
   StartZeit := GetTickCount64;
end;
```

Treffer anzeigen
```
procedure TForm1.Shape1MouseDown(Sender: TObject; Button: TMouseButton;
 Shift: TShiftState; X, Y: Integer);
begin
   inc(Treffer);
   Label1.caption:= 'Treffer: ' + IntToStr(Treffer);
end;
```

7 Zum Schluss musst du noch festlegen, dass die Prozedur „Start“ beim Einschalten des Timers gestartet wird.




8 Fertig, du kannst das Programm starten.

Natürlich bewegen sich die Bälle immer auf den gleichen Bahnen.
Zwei kleine Erweiterungen sorgt für etwas mehr Spielspass: Beim Programmstart, bei jedem Klick auf den grünen Ball und bei jedem „Fehlklick“ werden die Bälle per Zufall neu positioniert.

```
procedure TForm1.ZufallsPosition;
begin
  Randomize;
  for Zaehler := 1 to 5 do
  begin
    // verhindern, dass ein Ball oberhalb der Linie auftaucht
    while Ball[Zaehler].Top < Shape6.Top do
    begin
       Ball[Zaehler].Top := Random(Form1.Height - Ball[Zaehler].Width);
    end;
    Ball[Zaehler].Left := Random(Form1.Width - Ball[Zaehler].Width);
  end;
end;
```

Diese Prozedur wird beim Klick auf den Start-Button, beim Klick in die Form …

```
procedure TForm1.FormClick(Sender: TObject);
begin
   ZufallsPosition;
end;
```

… und beim Klick auf den grünen Ball ausgeführt.

```
procedure TForm1.Shape1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  inc(Treffer);
  Label1.caption:= 'Treffer: ' + IntToStr(Treffer);
  ZufallsPosition;
end;
```

Eine Ergänzung der Prozedur Start sorgt dafür, dass die Bälle an einer neuen Position „auftauchen“, wenn sie die Ränder erreichen:

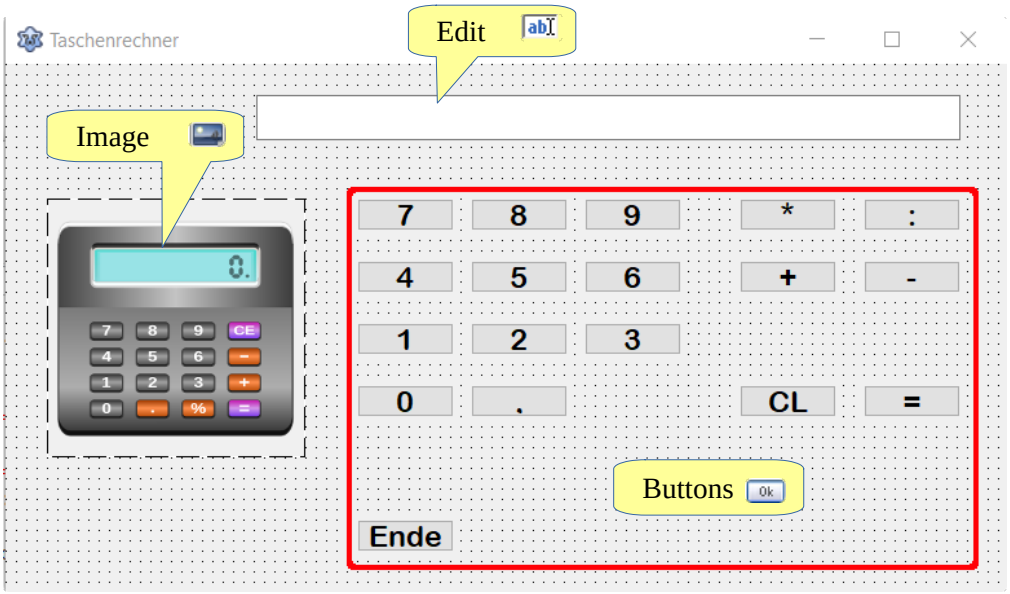
```
procedure TForm1.Start(Sender: TObject);
begin
  Label4.Caption:= FloatToStr(RoundTo((GetTickCount64 - StartZeit) / 1000, -1)) + ' s';
  for Zaehler := 1 to 5 do
  begin
    if Oben[Zaehler] = True then
    begin
       Ball[Zaehler].Top := Ball[Zaehler].Top - Geschwindigkeit
    end
    else
    begin
       Ball[Zaehler].Top := Ball[Zaehler].Top + Geschwindigkeit;
    end;
```

```
    if Rechts[Zaehler] = True then
    begin
      Ball[Zaehler].Left := Ball[Zaehler].Left + Geschwindigkeit
    end
    else
    begin
      Ball[Zaehler].Left := Ball[Zaehler].Left - Geschwindigkeit;
    end;

    // Ball hat obere Begrenzung nicht erreicht
    if Ball[Zaehler].Top <= 40 then
    begin
      Oben[Zaehler] := False;
      // neue Zufallsposition oben
      Ball[Zaehler].Left := Random(Form1.Width - Ball[Zaehler].Width);
    end;

    // Ball erreicht untere Grenze der Form (Form1.Height)
    // obere Kante (top) + Höhe des Balls (Height)
    if Ball[Zaehler].Top + Ball[Zaehler].Height >= Form1.Height then
    begin
      Oben[Zaehler] := True;
      // neue Zufallsposition unten
      Ball[Zaehler].Left := Random(Form1.Width - Ball[Zaehler].Width);
    end;

    // linke Kante des Balls (Left) ist <= 0 -> linke Grenze der Form
    if Ball[Zaehler].Left <= 0 then
    begin
      Rechts[Zaehler] := True;
      // neue Zufallsposition links
      Ball[Zaehler].Top := Random(Form1.Height - Ball[Zaehler].Height);
    end;

    // linke Kante des Balls (Left) + Breite des Balls (Width)
    // >= Breite der Form (Form1.Width)
    if Ball[Zaehler].Left + Ball[Zaehler].Width >= Form1.Width then
    begin
      Rechts[Zaehler] := False;
      // neue Zufallsposition rechts
      Ball[Zaehler].Top := Random(Form1.Height - Ball[Zaehler].Height);
    end;
  end;
end;
```

# Taschenrechner - Version mit Buttons

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Erstelle die Form.




2 Gib den Buttons mit den Zahlen und den Rechenzeichen unter Objektinspektor → Namen aussagekräftige Namen:
7 → Sieben
8 → Acht
. . .
+ → Plus
- → Minus
. . .
= → Ergebnis

3 Definiere unter `var` die beiden Zahlen:

```
var
  Zahl1: Double;
  Zahl2: Double;
```

4 Beginne mit der Prozedur Ergebnis.Click. Die Funktion Pos stellt die Position eines Zeichens oder eines Strings in einem anderen String fest und gibt die Position als Integer zurück:

```
Treffer:= Pos('ZuSuchendesZeichen', ZuDurchsuchenderString);
```

Die Funktion copy kann eine Teilstring aus einem anderen String herauslösen:

```
TeilString:= copy(VollständigerString, StartPosition, EndPosition);
```

```
procedure TForm1.ErgebnisClick(Sender: TObject);
var
  Rechnung: Double;
  Treffer: Integer;

// Summe
// nach + Zeichen suchen
Treffer:=  Pos('+', Edit1.Text);

// + Zeichen gefunden
if Treffer > 0  then
begin
   // das + Zeichen befindet sich nicht am Ende des Strings
   // Length(Edit1.Text)→ es wurde eine 2. Zahl eingegeben
   if Pos('+', Edit1.Text) < Length(Edit1.Text) then
   begin
      // Umwandlung nach Float
      // VollständigerString → Edit1.Text
      // StartPosition → 1. Zeichen nach +
      // EndPosition → Länge von Edit1.Text
      // ergibt die 2. Zahl
      Zahl2:= StrToFloat(copy(Edit1.Text,Treffer+1, Length(Edit1.Text)));
   end;
   // Berechnung der Summe
   Rechnung:= Zahl1 + Zahl2;
   // Zusammenstellen der gesamten Rechnung
   Edit1.Text:= Edit1.Text + ' = ' + FloatToStr(Rechnung);
end;
```

5 Wiederhole die Berechnungen für die Differenz und das Produkt.

Linux verwendet nicht das Komma, sondern stattdessen den Punkt als Dezimaltrennzeichen. Damit es nicht zu einer Fehlermeldung kommt, muss bei jeder Rechenoperation das Komma durch einen Punkt ersetzt werden.

```
Edit1.Text:= StringReplace(Edit1.Text, ',', '.', [rfReplaceAll, rfIgnoreCase]);
```

6 Bei der Berechnung des Quotienten ist eine Besonderheit zu beachten: Es darf nicht durch 0 dividiert werden.

```
// Quotient
// auf : untersuchen
Treffer:=  Pos(':',Edit1.Text);
if Treffer > 0  then
begin
   if pos(':', Edit1.Text) < Length(Edit1.Text) then
   begin
      Zahl2:= StrToFloat(copy(Edit1.Text,Treffer+1, Length(Edit1.Text)));
   end;
   try
      Rechnung:= Zahl1 / Zahl2;
      Edit1.Text:= Edit1.Text + ' = ' + FloatToStr(Rechnung);

      // Division durch 0 nicht möglich
      // Ausnahne →  EDivByZero
      Except
         on EDivByZero do
         begin
            Edit1.Text:= '';
            ShowMessage('Division durch 0 ist nicht möglich!');
         end;
   end;
end;
```

7 Natürlich kannst du die Reihenfolge der angeklickten Tasten nicht beeinflussen. Daher kann es passieren, dass eine Berechnung unvollständig ist, weil der = Button vorzeitig angeklickt wurde. Durch die Einführung einer Boolean-Variable kannst du das verhindern: Es wird erst dann gerechnet, wenn die Rechnung vollständig ist. Außerdem soll dann das Edit-Feld geleert werden. Füge unter `var` eine Variable hinzu:

```
GleichKlick: Boolean;
```

8 Ergänze die Prozedur ErgebnisClick

```
procedure TForm1.ErgebnisClick(Sender: TObject);
var
  Rechnung: Double;
  Treffer: Integer;
begin
   // Rechnung durch =-Klick beendet
   GleichKlick:= true;
   . . .
end;
```

9 Bei jedem Klick auf einen Button mit einer Zahl wird überprüft, ob die Rechnung bereits durchgeführt wurde.

```
procedure TForm1.EinsClick(Sender: TObject);
begin
   // bei erfolgter Rechnung Edit-Feld leeren
   // GleichKlick wurde bei der Rechnung auf true gesetzt
   // Rechnung wurde ausgeführt
   // → Textfeld leeren → neue Rechnung → GleichKlick false
   if GleichKlick = true then
   begin
      Edit1.text:= '';
      GleichKlick:= false;
   end;
   // die entsprechende Zahl ins Textfeld schreiben
   Edit1.Text:= Edit1.Text + Eins.Caption;
end;
```

Diese Prozedur musst du nun für jede Zahl erstellen.

10 Für den Klick auf das Komma genügt ein

```
Edit1.Text:= Edit1.Text + ',';
```

11 Bei einem Klick auf eine der Buttons mit den Rechenoperatoren gibt es das gleiche Problem wie beim = Button: Das Programm muss feststellen, ob die Rechnung schon vollständig ist. Hier kannst du wieder die Variable GleichKlick verwenden.

```
procedure TForm1.PlusClick(Sender: TObject);
var
  Treffer: Integer;
begin
   // bei erfolgter Rechnung Edit-Feld leeren
   // GleichKlick wurde bei der Rechnung auf true gesetzt
   // Rechnung wurde ausgeführt
   // → Textfeld leeren → neue Rechnung → GleichKlick false
   if GleichKlick= true then
   begin
      Edit1.text:= '';
      GleichKlick:= false;
   end
   // eingegebene Zahl ermitteln
   else
   begin
      try
         Edit1.Text:= Edit1.Text + ' + ';
         // Position des + Zeichens feststellen
         // wenn vor dem + Zeichen keine Zahl steht
         // ist eine Umwandlung in eine Zahl nicht möglich
         // weiter mit Except
         Treffer:=  Pos('+', Edit1.Text);
         Zahl1:= StrToFloat(copy(Edit1.Text,0, Treffer-1));
```

```
            // Except → Textfeld leeren
            // Fehler anzeigen
            ShowMessage('Du musst eine Zahl eingeben!');
            Edit1.Text:= '';
         end;
      end;
   end;
```

12 Diese Prozedur musst du für alle Rechenoperation erstellen.
Achte darauf, dass du die richtige Rechenoperation wählst und nach dem richtigen Rechenzeichen suchen lässt.

13 Fertig, das Programm ist startbereit.

Bei der Bedienung des Programms gibt es noch viele Fehlerquellen. Eine Erweiterung des Komma-Klicks sorgt dafür, dass in einer Zahl nicht mehrere Kommas eingeben werden können:

```
procedure TForm1.KommaClick(Sender: TObject);
var
  Treffer, TrefferKomma1, TrefferKomma2: Integer;
begin
   Edit1.Text:= Edit1.Text + ',';
   Treffer:=  Pos(' ',Edit1.Text);
   // noch keine Leerstelle → erste Zahl
   if Treffer = 0 then
   begin
      // erstes Komma suchen
      TrefferKomma1 := Pos(',', Edit1.Text);
      // nach einem weiteren Komma suchen
      // beginnend mit der Position des ersten Kommas (PosEx)
      // unter uses muss StrUtils hinzugefügt werden
      TrefferKomma2 := PosEx(',', Edit1.Text, TrefferKomma1 + 1);
      // zweites Komma gefunden
      // letztes Zeichen abschneiden → Length(Edit1.Text) - 1
      if TrefferKomma2 > 0 then
         Edit1.Text:=copy(Edit1.Text, 0, Length(Edit1.Text) - 1);
   end
   // es gibt eine Leerstelle → zweite Zahl
   else
   begin
      // String ab dem Leerzeichen nach Komma durchsuchen
      TrefferKomma1 := PosEx(',', Edit1.Text, Treffer);
      TrefferKomma2 := PosEx(',', Edit1.Text, TrefferKomma1 + 1);
      // zweites Komma gefunden
      // letztes Zeichen abschneiden → Length(Edit1.Text) - 1
      if TrefferKomma2 > 0 then
         Edit1.Text:= copy(Edit1.Text, 0, Length(Edit1.Text) - 1);
    end;
end;
```

Es können mehrere Rechenoperation eingegeben werden, oder die zweite Zahl fehlt. Dieses Problem kann jeweils durch einen `try .. except`-Block gelöst werden.
Am Beispiel der Summenberechnung:

```
Treffer:=  Pos('+',Edit1.Text);
if Treffer > 0  then
begin
   if pos('+', Edit1.Text) < Length(Edit1.Text) -1 then
   begin
      try
         Zahl2:= StrToFloat(copy(Edit1.Text,Treffer + 1, Length(Edit1.Text)));
         Rechnung:= Zahl1 + Zahl2;
         Edit1.Text:= Edit1.Text + ' = ' + FloatToStr(Rechnung);
         GleichKlick:= true;
      except
         ShowMessage('Die Rechnung ist nicht vollständig!');
      end;
   end;
end;
```

Hier findest du noch einen erweiterten Taschenrechner.

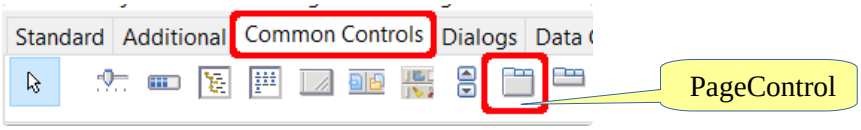
# Platz sparen mit PageControl und Tabs

Statt mehrere Fenster aufzumachen kannst du auch platzsparend mit Tabs in einem Fenster arbeiten.

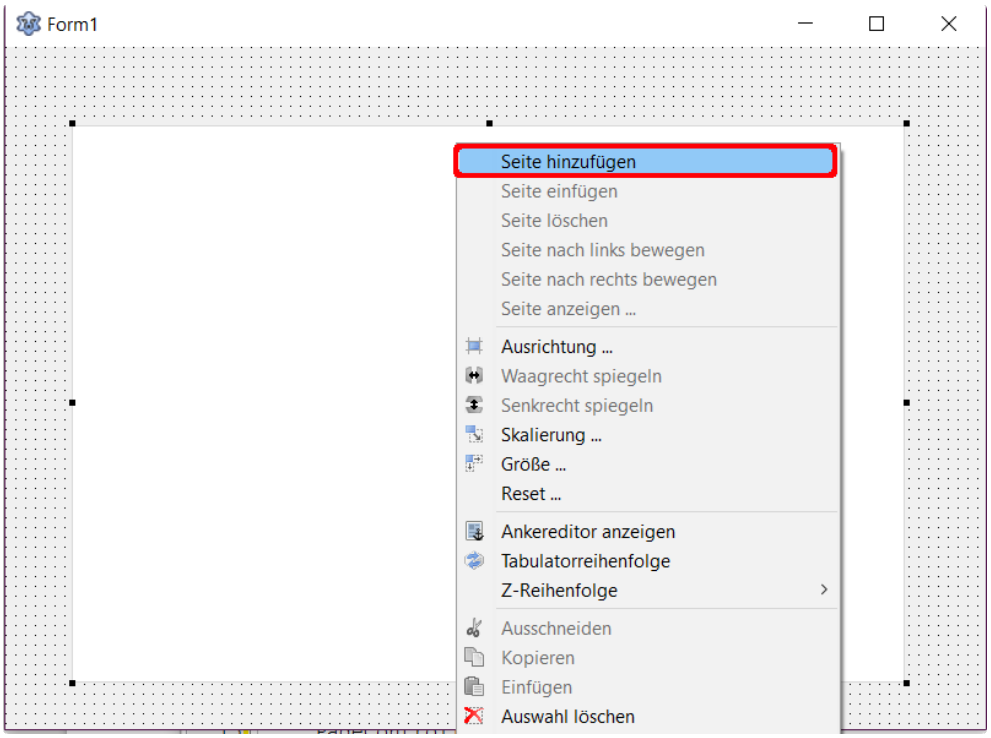
## Volumenberechnung

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

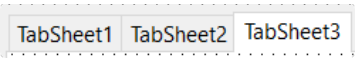
1 Erstelle die Form. Setze als Erstes ein PageControl in die Form.



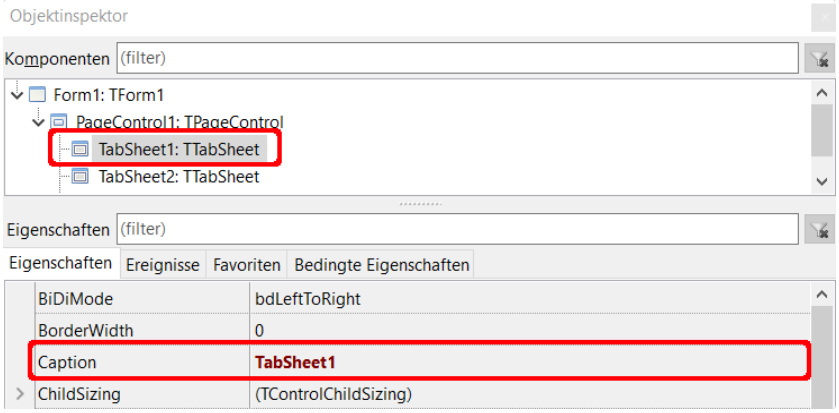
2 Füge drei Seiten hinzu:



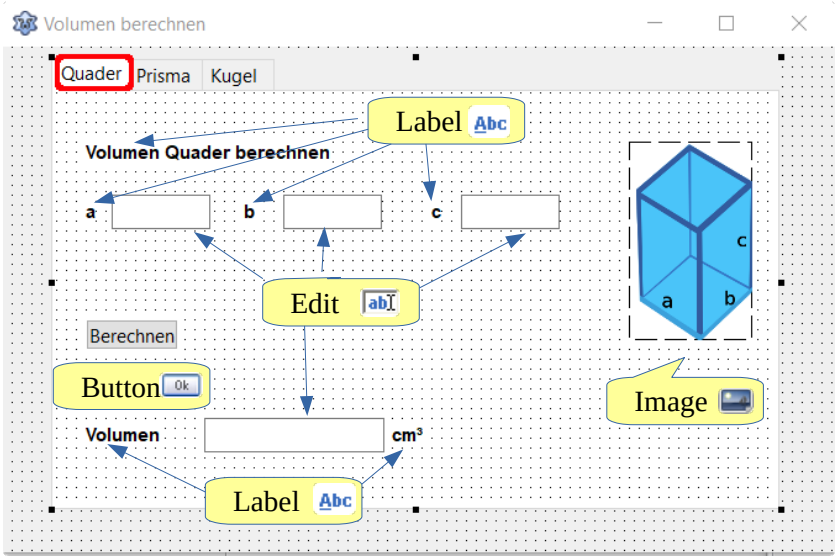

So muss es aussehen:




Ändere die Eigenschaft Caption der TabSheets:
TabSheet1 → Quader
TabSheet2 → Prisma
TabSheet3 → Kugel



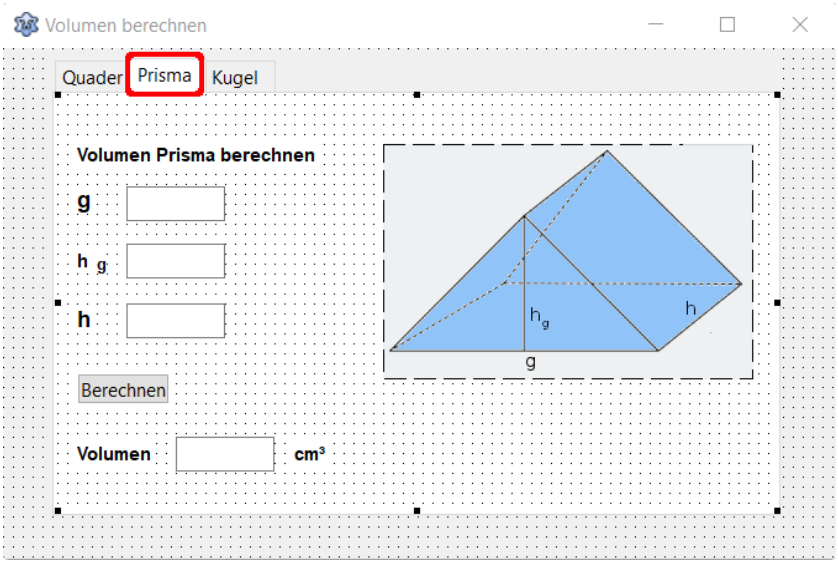
3 Erstelle zu den einzelnen Tabs jeweils ein Formular.
Das Bild findest du hier.




Die Berechnung des Volumens eines Quaders:

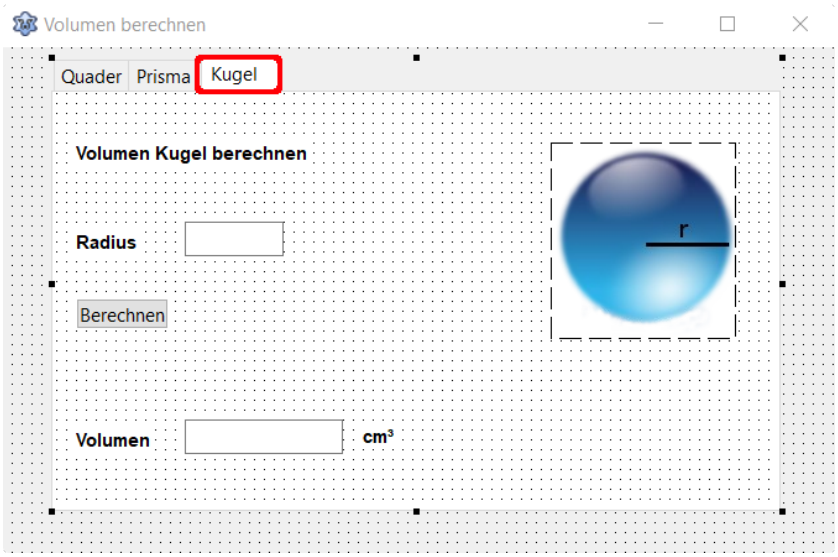
```
procedure TForm1.QuaderBerechnenClick(Sender: TObject);
var
  Volumen: Double;
begin
   // EingabeQuadera → Eingabefeld a
   // EingabeQuaderb → Eingabefeld b
   // EingabeQuaderc → Eingabefeld c
   // V = a * b * c
   // vor der Rechnung → nach Float umwandeln
   Volumen:= StrToFloat(EingabeQuadera.Text) *
   StrToFloat(EingabeQuaderb.Text) * StrToFloat(EingabeQuaderc.Text);
   Edit4.Text:= FloatToStr(Volumen);
end;
```

Das Bild findest du hier.




Die Berechnung:
$V = g * h_g / 2 * h$

Das Bild findest du hier.



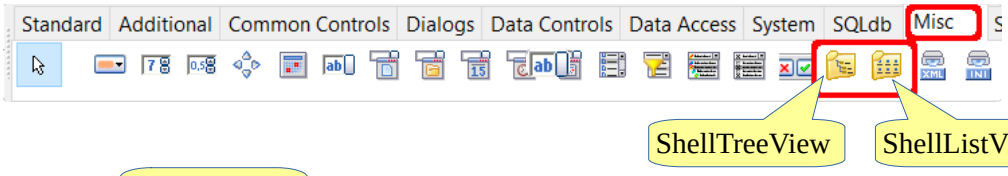

Die Berechnung:
V = 4/3 * Pi * r * r * r
Lazarus kennt Pi als Konstante.

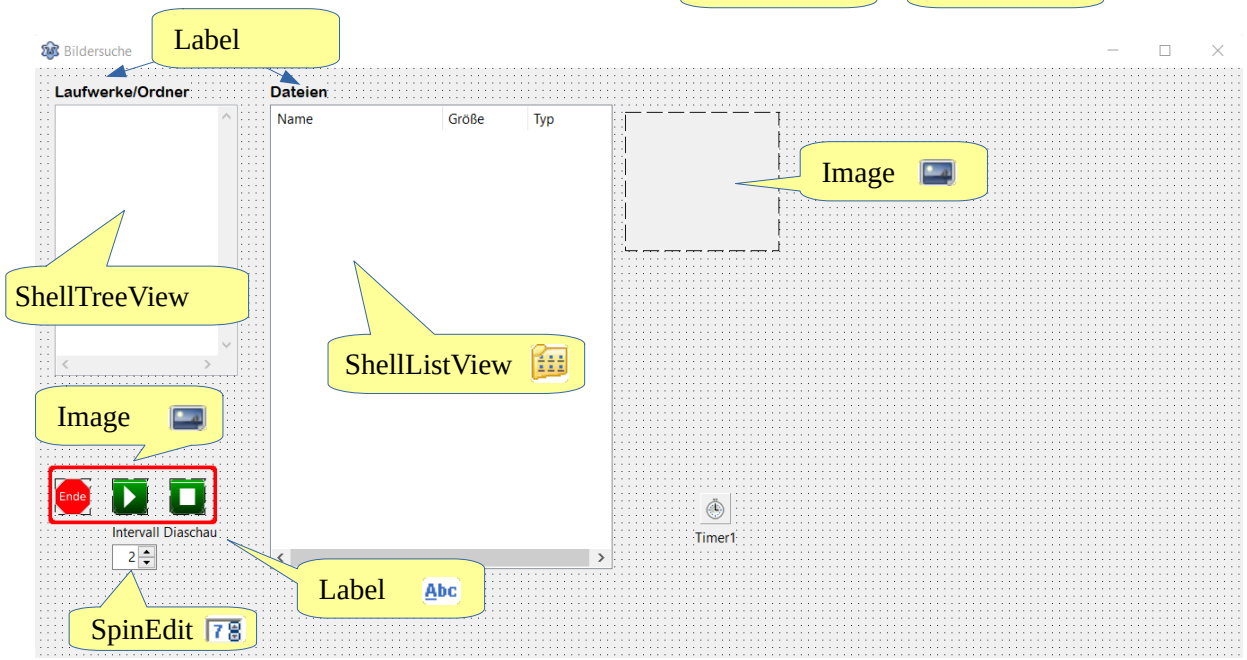
# Laufwerke und Ordner anzeigen

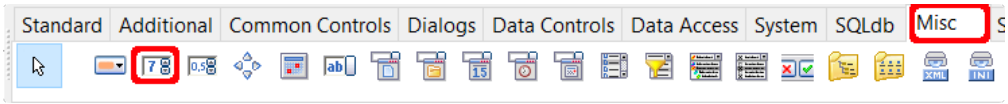
## Bilder suchen und anzeigen

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

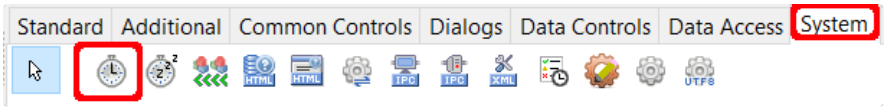
1 Erstelle die Form.




Den Timer findest du hier:

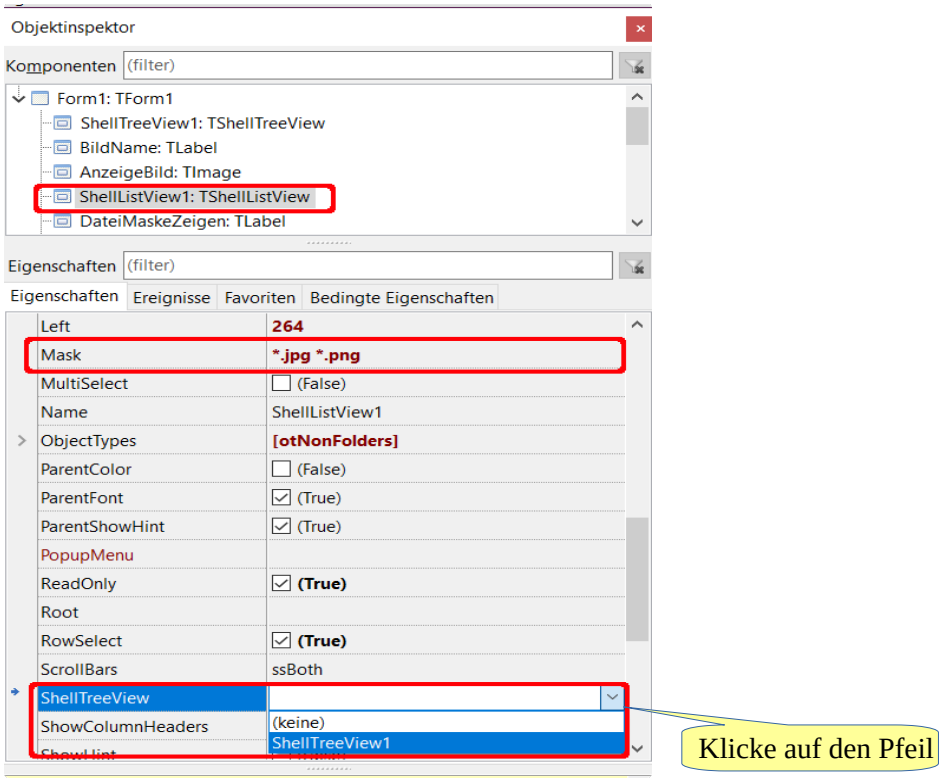



Setze die Eigenschaft Visible der Objekte SpinEdit1 und des Labels „Intervall Diaschau“ im Objektinspektor auf false. Diese Elemente sollen erst beim Start der Diaschau sichtbar werden.

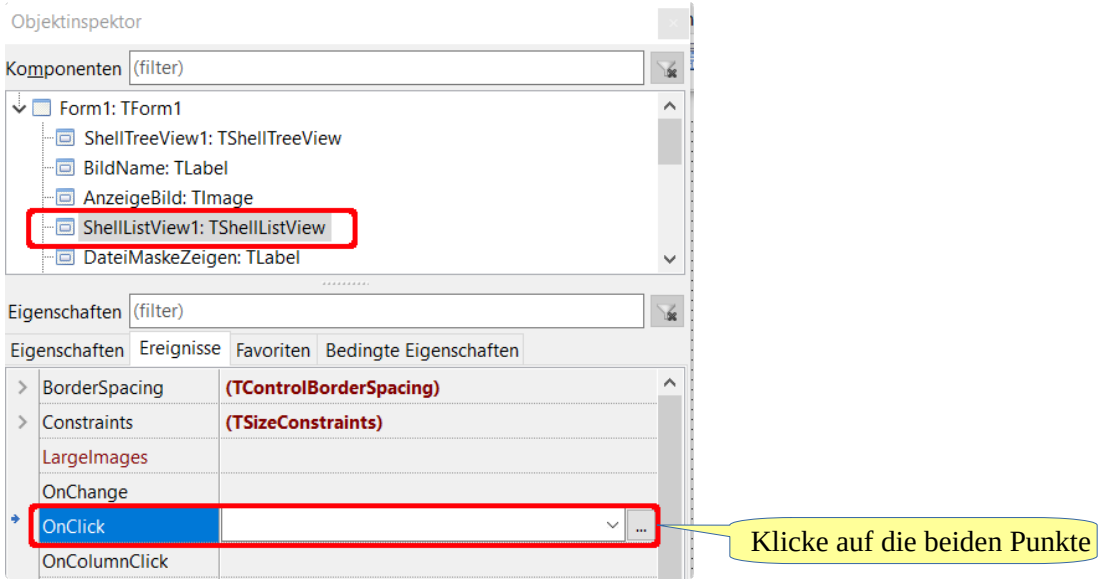
2 Setze die Eigenschaften der ShellListView:

**Mask:** Endungen der Dateien, die angezeigt werden sollen

**ShellTreeview:** Verknüpfung zu ShellTreeView, die Dateien des dort angeklickten Ordners werden in der ShellListView angezeigt




2 Erstelle die Prozedur ShellListView1Click:



```
procedure TForm1.ShellListView1Click(Sender: TObject);
begin
   // testen, ob  ShellListView1.Selected existiert
   // also nicht Nichts (Nil) ist
   if ShellListView1.Selected <> Nil then
   begin
      // Umwandlung des Dateinamens nach Kleinbuchstaben -> LowerCase
      // ShellListView1.Selected.caption → Dateiname
      // auch JPG/PNG werden dann erfasst
      // nur wenn Dateiendung korrekt ist wird das Bild angezeigt
      if (Pos('.jpg', LowerCase(ShellListView1.Selected.caption)) > 0) or
     (Pos('.png', LowerCase(ShellListView1.Selected.caption)) > 0) then
      begin
         // Anzeige des Dateinamens
         BildName.Caption:=ShellListView1.Selected.caption;
         // wenn Bildformat ungültig -> weiter mit except
         try
            // Bildeigenschaften festlegen
            AnzeigeBild.Width:= 600;
            AnzeigeBild.Height:= 400;
            AnzeigeBild.Proportional:= true;
            AnzeigeBild.Stretch:= true;
            // Bild anzeigen
            // ShellListView1.GetPathFromItem(ShellListView1.Selected)
            // ist der vollständige Pfad und Dateiname des
            // angeklickten Eintrags
            AnzeigeBild.Picture.LoadFromFile(ShellListView1.
            GetPathFromItem(ShellListView1. Selected));
         except
            ShowMessage('Keine Bilddatei');
         end;
      end;
   end;
end;
```

2 Jetzt kommt die Programmierung der Diaschau an die Reihe. Die Bilder findest du hier und hier.

Erstelle mit einem Doppelklick auf den Start-Button die Klick-Prozedur:

```
procedure TForm1.StartDiaschauClick(Sender: TObject);
begin
   // Timer starten
   Timer1.Enabled:= true;
   // Intervall der Diaschau anzeigen
   ZeitSekunden.Visible:= true;
   // Anzeige der Intervall-Auswahl
   SpinEdit1.Visible:= true;
end;
```

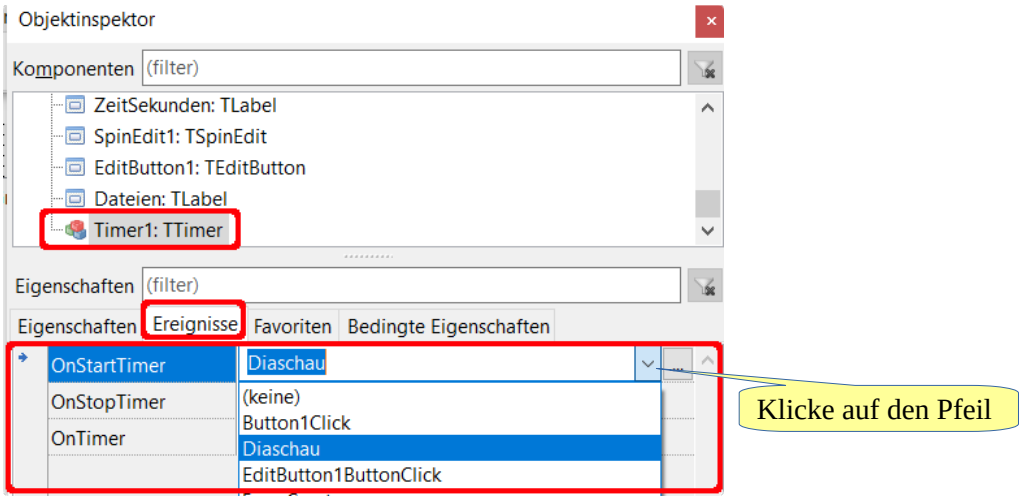
Die Klick-Prozedur des Stopp-Buttons stoppt den Timer und macht das Label und die Auswahl des Intervalls „unsichtbar“

```
procedure TForm1.StopDiaschauClick(Sender: TObject);
begin
   Timer1.Enabled:= false;
   ZeitSekunden.Visible:= false;
   SpinEdit1.Visible:= false;
end;

procedure TForm1.Diaschau(Sender: Tobject);
// IndexListe ist die aktuelle Position in der Listbox
var
  IndexListe: Integer;

begin
   // ShellListView1.ItemIndex ist der gerade markierte Eintrag
   IndexListe:= ShellListView1.ItemIndex;

   // Diaschau soll solange laufen, bis der letzte Eintrag der Liste
   // erreicht ist →  ShellListView1.Items.Count
   if IndexListe < ShellListView1.Items.Count - 1 then
   begin
      // nach Erhöhen um 1 der IndexListe wird auch der markierte Eintrag
      // der Liste um 1 erhöht
      inc(IndexListe);
      ShellListView1.ItemIndex:= IndexListe;
      // Umwandlung des Dateinamens nach Kleinbuchstaben -> LowerCase
      // ShellListView1.Selected.caption → Dateiname
      // auch JPG/PNG werden dann erfasst
      // nur wenn Dateiendung korrekt ist wird das Bild angezeigt
      if (Pos('.jpg', LowerCase(ShellListView1.Selected.caption)) > 0)
      or (Pos('.png', LowerCase(ShellListView1.Selected.caption)) > 0)
      then
      begin
         // Bildeigenschaften festlegen
         AnzeigeBild.Width:= 600;
         AnzeigeBild.Height:= 400;
         AnzeigeBild.Proportional:= true;
         AnzeigeBild.Stretch:= true;
         // Bild anzeigen
         // ShellListView1.GetPathFromItem(ShellListView1.Selected)
         // ist der vollständige Pfad und Dateiname des
         // angeklickten Eintrags
         AnzeigeBild.Picture.LoadFromFile(ShellListView1.
        GetPathFromItem(ShellListView1.Selected));
      end;
   end;
end;
```

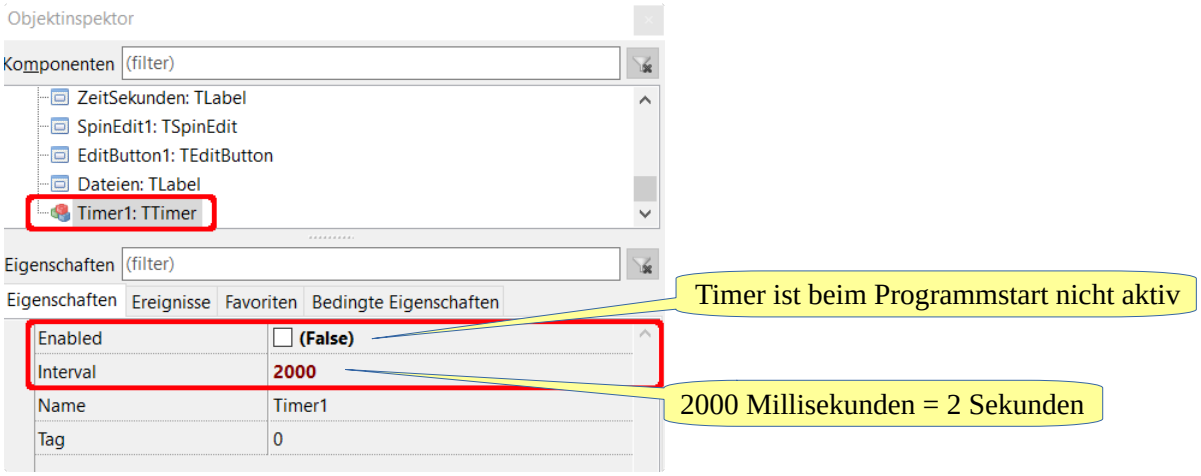
Da du die Prozedur selbst erstellt hast, wurde sie nicht automatisch „angemeldet“. Du musst das selbst unter `type` erledigen:

```
procedure Diaschau(Sender: Tobject);
```

3 Lege beim Timer1 unter Ereignisse den Start der Prozedur Diaschau bei Aktivieren des Timers fest.



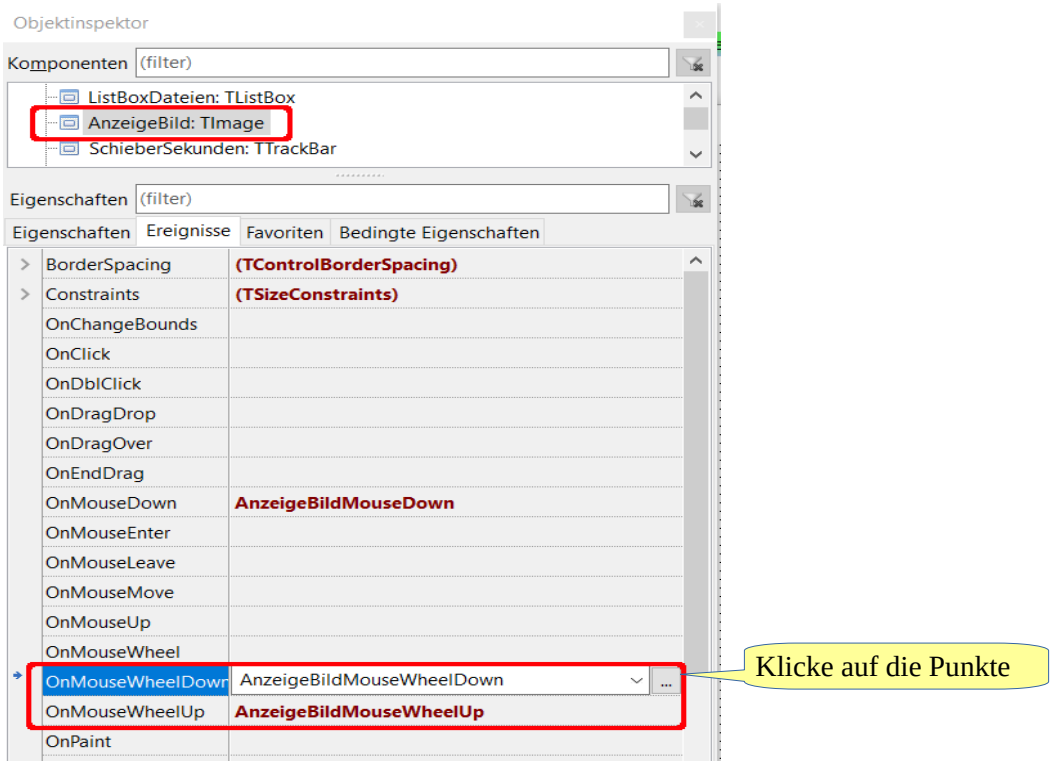

4 Du musst bei den Eigenschaften des Timers noch seine Laufzeit (Intervall) beim Start festlegen und die Eigenschaft Enabled muss false sein:




5 Ein Doppelklick auf SpinEdit1 erstellt die Prozedur `SpinEdit1Change`:

```
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
   Timer1.Enabled:= false;
   Timer1.Interval:= SpinEdit1.Value*1000;
   // den Focus zurück auf die Dateiliste setzen
   ShellListView1.SetFocus;
   Timer1.Enabled:= true;
end;
```

5 Wenn das Mausrad nach oben oder nach unten bewegt wird, soll das Bild vergrößert/verkleinert werden.
Erstelle im Objektinspektor die Prozeduren `AnzeigeBildMouseWheelDown` und `AnzeigeBildMouseWheelUp:`




```
procedure TForm1.AnzeigeBildMouseWheelDown(Sender: TObject; Shift:
TshiftState; MousePos: TPoint; var Handled: Boolean);
begin
   if AnzeigeBild.Width> 200 then
   begin
      AnzeigeBild.Width:=AnzeigeBild.Width - 10;
   end;
   // aktuelle Bildbreite sichern
   BildBreite:= AnzeigeBild.Width;
end;

procedure TForm1.AnzeigeBildMouseWheelUp(Sender: TObject; Shift:
TshiftState; MousePos: TPoint; var Handled: Boolean);
begin
   if AnzeigeBild.Width < 800 then
   begin
      AnzeigeBild.Width:= AnzeigeBild.Width + 10;
   end;
   // aktuelle Bildbreite sichern
   BildBreite:= AnzeigeBild.Width;
end;
```

6 Fertig, das Programm ist startbereit.

Unter Windows ist es möglich mit einem Rechtsklick Standardprogramme zu öffnen:
Du musst unter `uses` die Bibliothek `ShellAPI` hinzufügen.

```
procedure TForm1.AnzeigeBildMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
   // rechte Maustaste gedrückt (mbRight)
   if (Button = mbRight) then
   begin
      if ShellExecute(0, 'open', PChar('rundll32.exe'),
         PChar('shell32.dll,OpenAs_RunDLL ' +
         ShellListView1.GetPathFromItem(ShellListView1.Selected)), nil,
         SW_SHOWNORMAL) <= 32 then ShowMessage('Es ist ein Fehler aufgetreten');
   end;
end;
```

Wenn du möchtest, dass zu den Bildern ein Tipp angezeigt wird, musst du eine Prozedur `FormCreate` erstellen:

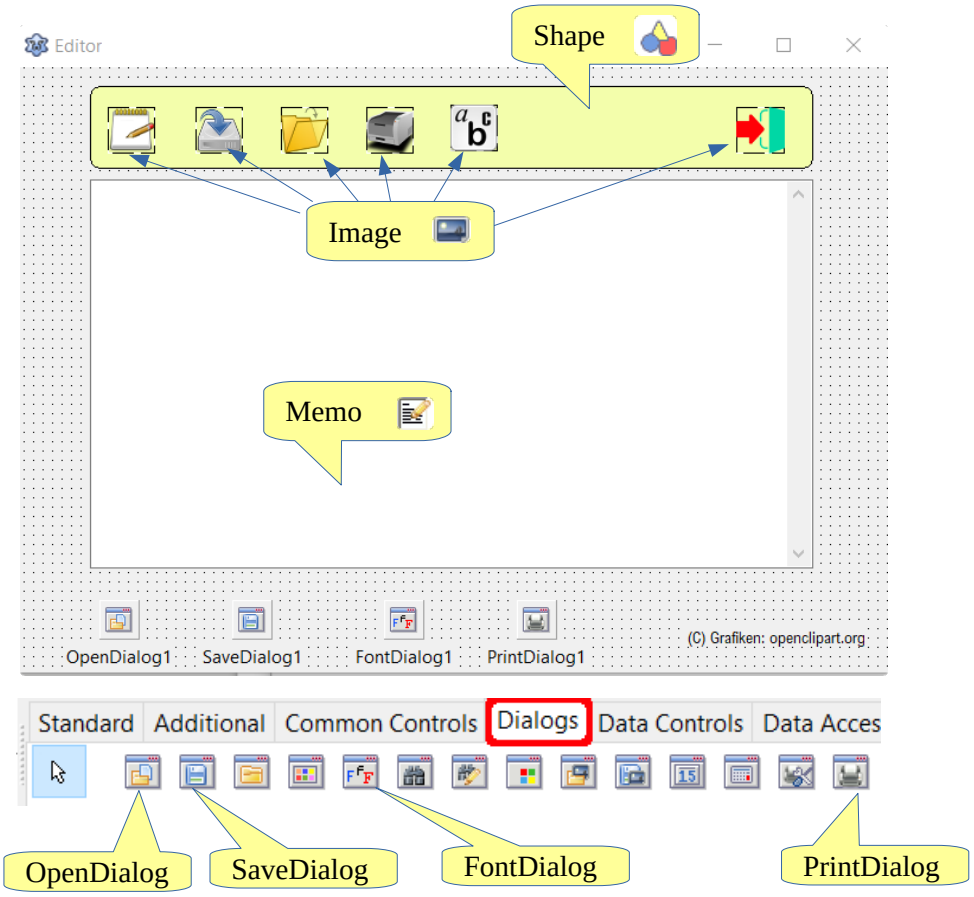
```
procedure TForm1.FormCreate(Sender: TObject);
begin
   // Spaltenüberschriften in ShellListView auf Deutsch
   ShellListView1.Column[1].Caption := 'Größe';
   ShellListView1.Column[2].Caption := 'Typ';

   // Hinweis Mausrad/Öffnen mit ...
   AnzeigeBild.ShowHint:= true;
   AnzeigeBild.Hint:= 'Mausrad → Größe ändern' + sLineBreak + 'Rechte Maustaste
   Öffnen mit ...';

   // Hinweise zu den Bilderbuttons
   StartDiaschau.Hint:= 'Diaschau starten';
   StartDiaschau.ShowHint:= true;
   StopDiaschau.Hint:= 'Diaschau stoppen';
   StopDiaschau.ShowHint:= true;
   ProgrammEnde.Hint:= 'Programm beenden';
   ProgrammEnde.ShowHint:= true;
end;
```

# Texte erstellen, speichern, laden und drucken

## Einfacher Texteditor

Du kannst das Programm hier herunterladen.

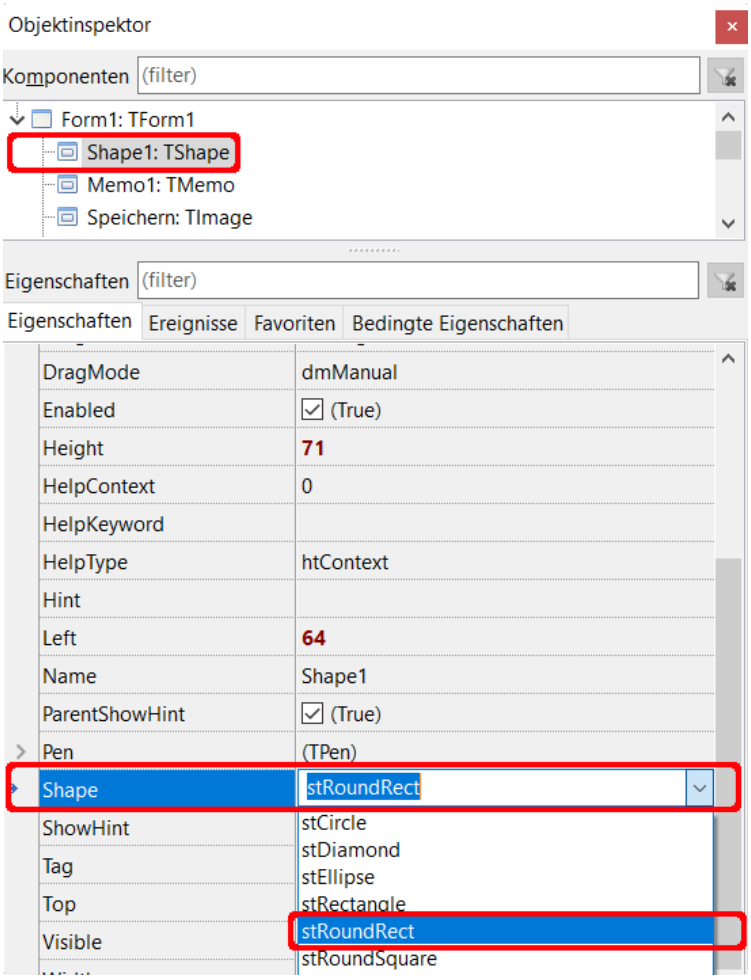
1 Erstelle die Form.




2 Die Bilder findest du hier.

neuer_text.png, speichern.png, oeffnen.png, drucken.png, schrift.png und exit.exit.png

3 Als Shape wählst du stRoundRect, unter Brush → Color kannst du eine Farbe deiner Wahl einstellen.




4 Damit die Prozeduren besser unterschieden werden können, wähle im Objektinspektor → Name aussagekräftige Namen für die Bilder:


5 Die Prozedur `NeuerTextClick` löscht einfach den gesamten Text des Memos

```
procedure TForm1.NeuerTextClick(Sender: TObject);
begin
   Memo1.clear;
end;
```

6

**Datei öffnen:**

```
procedure TForm1.OeffnenClick(Sender: TObject);
begin
   // Filter für die Anzeige festlegen (*.txt)
   OpenDialog1.Filter := 'Texte|*.txt';
   // Titel des Dialogs
   OpenDialog1.Title:= 'Datei öffnen ...';
   // angeklickte Datei öffnen
   if OpenDialog1.Execute
   then Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

**Datei speichern:**

```
procedure TForm1.SpeichernClick(Sender: TObject);
begin
   // Filter für die Anzeige festlegen
   SaveDialog1.Filter := 'Texte|*.txt';
   // Endung txt wird automatisch angehängt
   SaveDialog1.DefaultExt:='txt';
   // Titel des Dialogs
   SaveDialog1.Title:= 'Datei speichern als ...';
   // vollständigen Inhalt des Memos speichern
   if SaveDialog1.Execute
   then Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

**Schriftart wählen:**

```
procedure TForm1.SchriftartClick(Sender: TObject);
var
  Zaehler: Integer;
begin
   if FontDialog1.Execute then
   begin
      // Titel des Dialogs
      FontDialog1.Title:= 'Schriftart wählen ...';
      // Standard-Schrift bestimmen
      FontDialog1.Font.Name:= 'Arial';
      // gewählte Schriftart übernehmen
      Memo1.Font:= FontDialog1.Font;
   end;
end;
```

**Drucken:**

```
procedure TForm1.DruckenClick(Sender: TObject);
var
  Zaehler, LinkerRand, ZeilenAbstand: Integer;
  SchriftInPixel: Double;
begin
   PrintDialog1.Title:= 'Datei drucken ...';
   If PrintDialog1.Execute then
   begin
      // Schrifart aus Memo übernehmen
      Printer.Canvas.Font := FontDialog1.Font;
      // Druck starten
      Printer.BeginDoc;
      // 5 % oberer Rand
      ZeilenAbstand:= Round(Printer.PageHeight * 0.05);
      // 10 % linker Rand
      LinkerRand:= Round(Printer.PageWidth * 0.1);
      // Schriftgröße in Pixel umrechnen
      SchriftInPixel:= Printer.PageHeight/1000 * FontDialog1.Font.Size * 1.33;
      // zeilenweise bis zur letzten Zeile drucken
      for Zaehler:=0 to Memo1.Lines.Count-1 do
      begin
          Printer.Canvas.TextOut(LinkerRand, ZeilenAbstand, Memo1.Lines[Zaehler]);
           // Zeilenabstand entsprechend der Schriftgröße
          ZeilenAbstand:= ZeilenAbstand + Round(SchriftInPixel);
          // Zeilenabstand > Höhe der Seite - 5% -> neue Seite
          if ZeilenAbstand > Printer.PageHeight - Round(Printer.PageHeight * 0.05) then
          begin
             Printer.NewPage;
             ZeilenAbstand:= Round(Printer.PageHeight * 0.05);
          end;
      end;
   // Druck beenden
   Printer.EndDoc;
   end;
end;
```

**Unter uses musst du die Bibliothek Printers hinzufügen.**

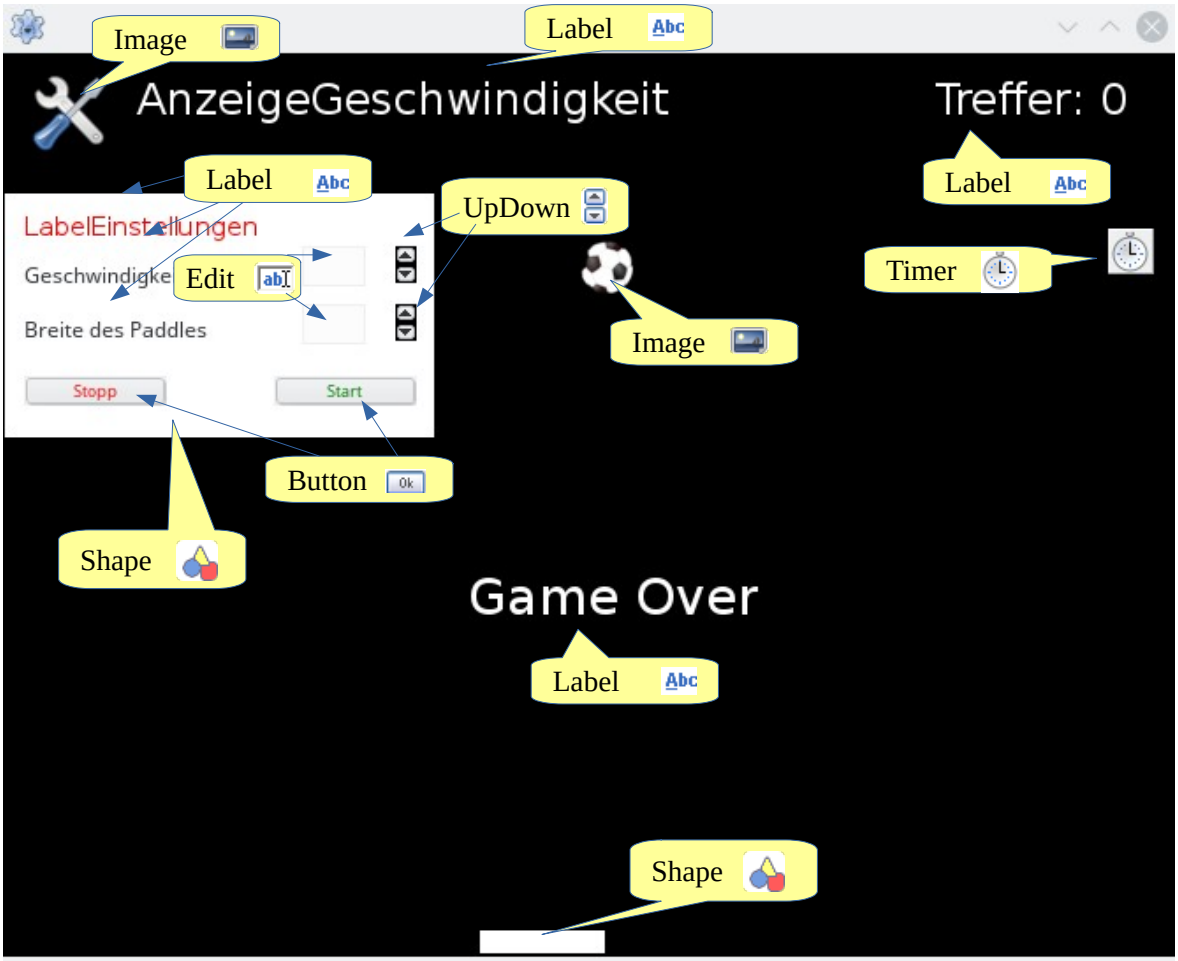
7 Das Programm ist startbereit.

# Projekte

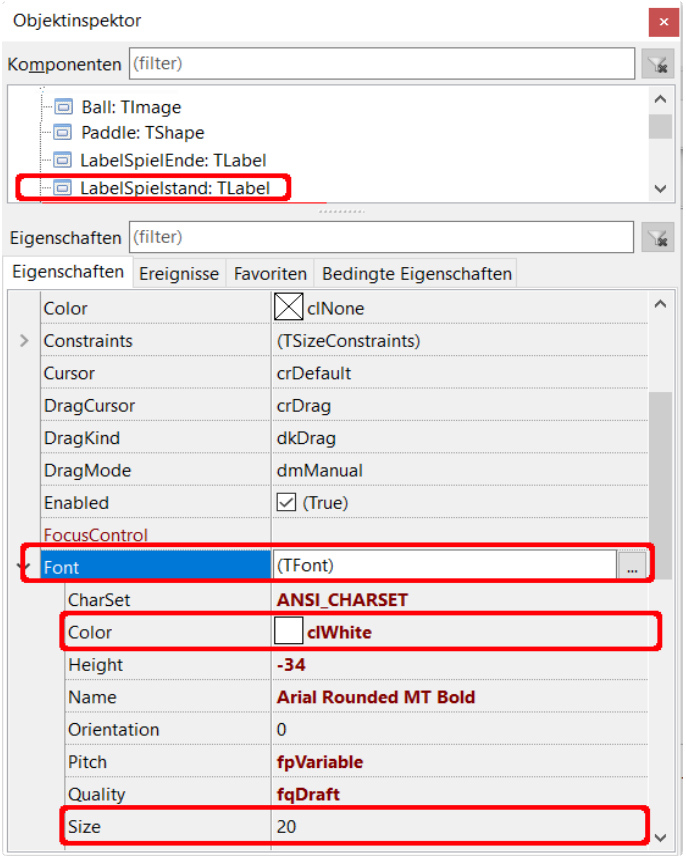
## Pong-Spiel

Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

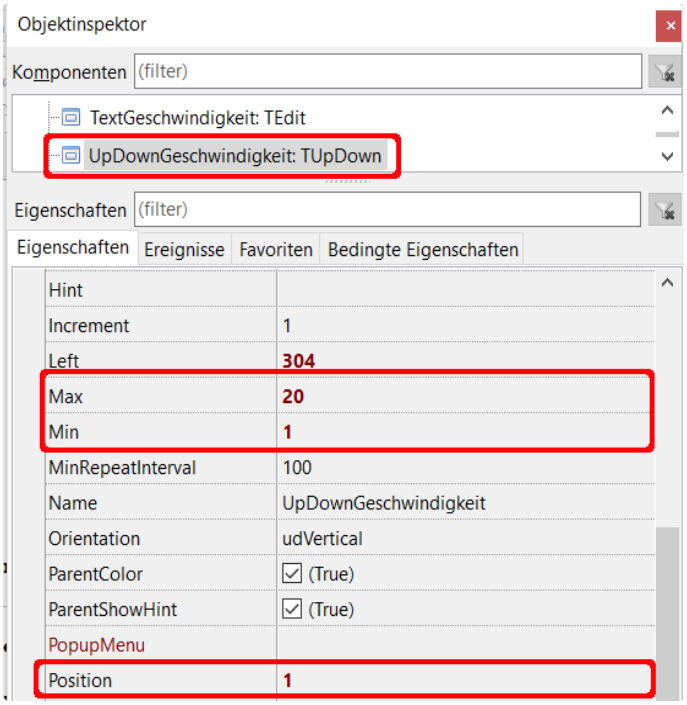
1 Erstelle die Form.



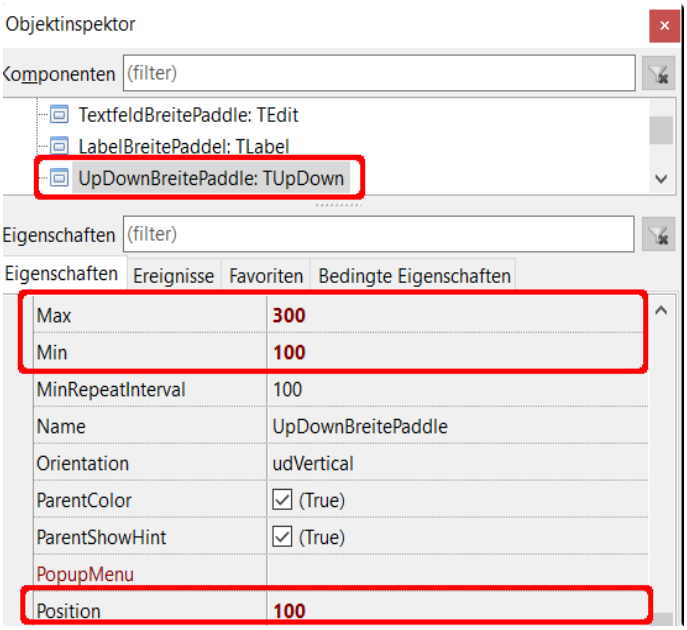

2 Lege die Eigenschaften der Labels „Game Over“, „AnzeigeGeschwindigkeit“ und „Treffer“ fest:



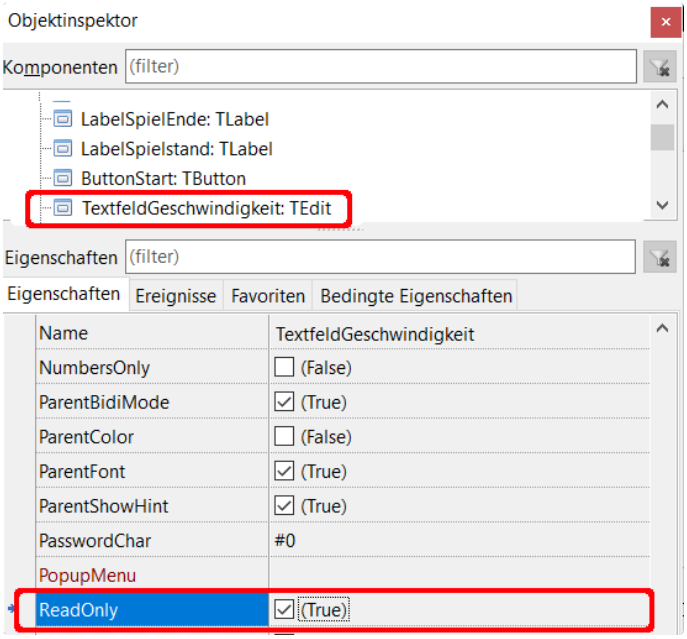

3 Die Eigenschaften des UpDown-Elements für die Geschwindigkeit:



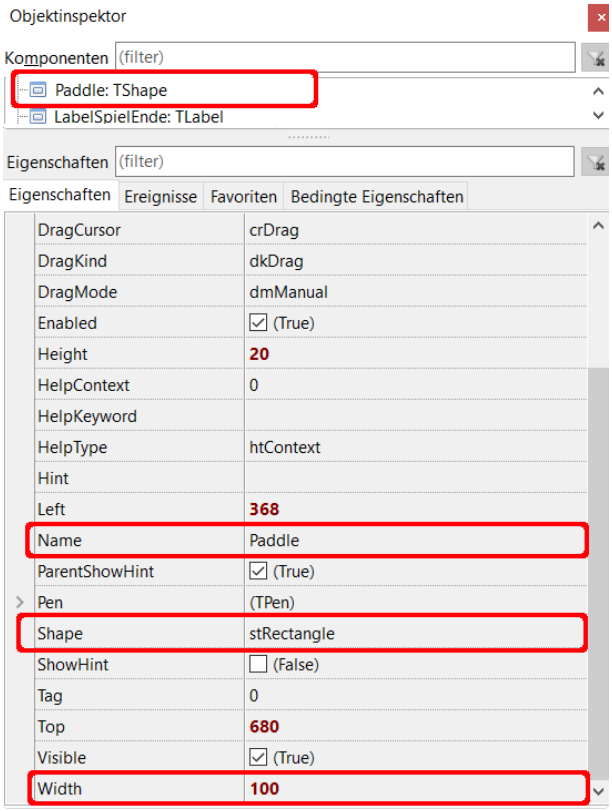
4 Die Eigenschaften des UpDown-Elements für die Breite des Paddles:



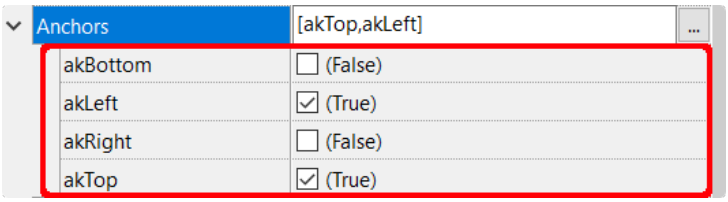

5 Setze die Eigenschaft ReadOnly der beiden Edit-Felder auf true.



6 Eigenschaften Paddle:

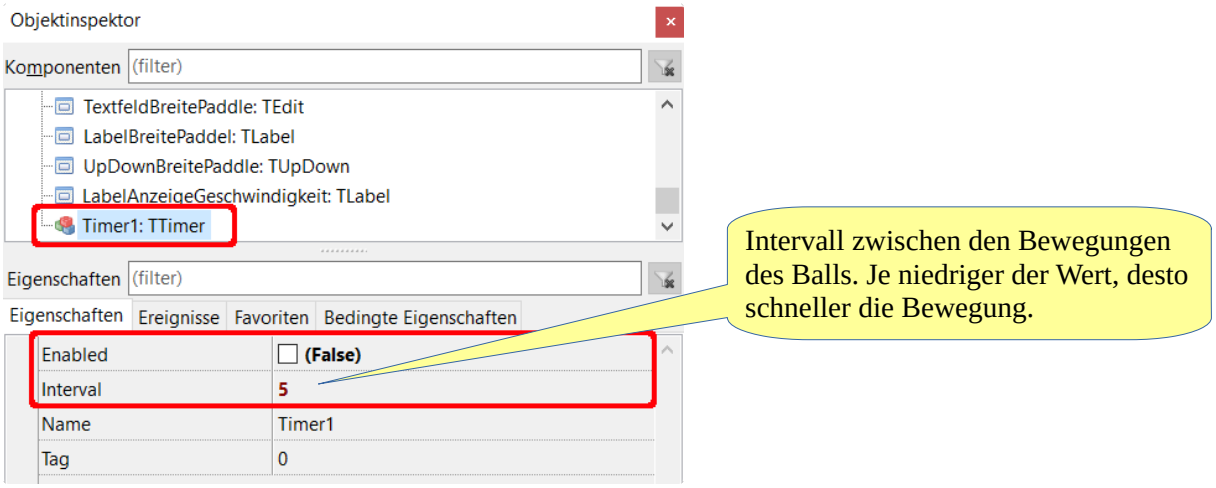



Kontrolliere, ob die Anker richtig gesetzt sind. Sie sorgen dafür, dass die Elemente auch nach einer Änderung der Formgröße an ihrem Platz bleiben. Das Paddle bleibt immer am unteren Rand.

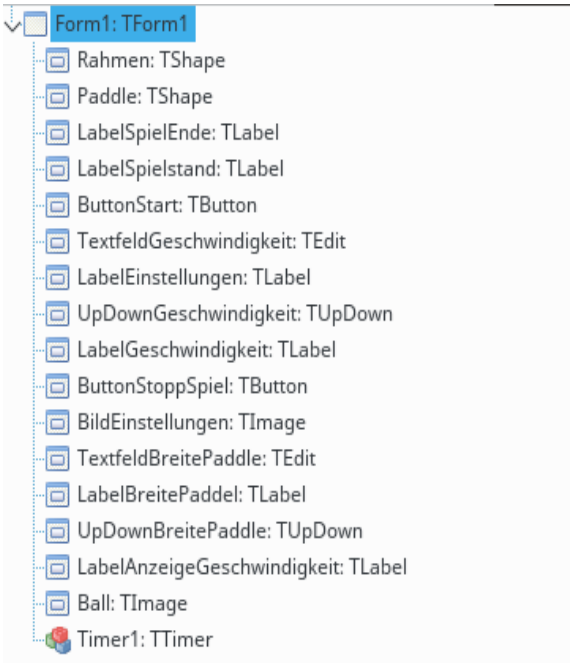



7 Setze das Bild in die Form.

8 Die Eigenschaften des Timers:




Alle Elemente der Form:




9 Jetzt kann die Programmierung beginnen. Definiere zunächst Variable:

```
Treffer: Integer;
// Geschwindigkeit X-Achse, Geschwindigkeit Y-Achse
GeschwindigkeitX, GeschwindigkeitY: Integer;
```

10 Als nächstes sollen die Einstellungen programmiert werden.
Ein Doppelklick auf das UpDown-Element Geschwindigkeit erstellt die Standardprozedur.

```
procedure TForm1.UpDownGeschwindigkeitClick(Sender: TObject; Button:
TUDBtnType);
begin
   // Geschwindigkeit X-Achse erhöhen
   // Geschwindigkeit Y-Achse erhöhen
   inc(GeschwindigkeitX);
   inc(GeschwindigkeitY);

   // Position des UpDown-Elements Geschwindigkeit in das Textfeld übertragen
   TextfeldGeschwindigkeit.Text:= IntToStr(UpDownGeschwindigkeit.Position);

   // Position des UpDown-Elements Geschwindigkeit in das Label übertragen
   AnzeigeGeschwindigkeit.Caption:= 'Geschwindigkeit: ' +
   IntToStr(UpDownGeschwindigkeit.Position);
end;
```

Ein Doppelklick auf das UpDown-Element Paddle-Breite erstellt die Standardprozedur.

```
procedure TForm1.UpDownBreitePaddleClick(Sender: TObject; Button:
TUDBtnType);
begin
   // Position des UpDown-Elements Paddle-Breite in das Textfeld
   // übertragen
   TexteBreitePaddle.Text:= IntToStr(UpDownBreitePaddle.Position);

   // Paddle in der eingestellten Breite darstellen
   Paddle.Width:= UpDownBreitePaddle.Position;
end;
```

11 Erstelle eine Prozedur SpielStart, sie soll später mit einem Klick auf den Button Start ausgeführt werden. Da du sie von Hand erstellt hast, musst du sie in der Liste der Prozeduren am Anfang des Programms hinzufügen.

```
procedure SpielStart;
```

```
procedure TForm1.SpielStart;
begin
   // zufällige Startposition > Hälfte der Höhe
   Randomize;
   Ball.Left:= Random(Form1.Width - Ball.Height);
   Ball.Top:= Random((Form1.Height - Ball.Width) div 2);
   Treffer:= 0;
   // Elemente Fenster Einstellungen ausblenden
   // Labels
   LabelSpielEnde.Visible:= false;
   LabelEinstellungen.Visible:= false;
   LabelGeschwindigkeit.Visible:= false;
   LabelBreitePaddel.Visible:= false;
   LabelBreitePaddel.Visible:= false;
   // Buttons
   ButtonStart.Visible:= false;
   ButtonStoppSpiel.Visible:= false;
   // UpDown-Elemente
   UpDownGeschwindigkeit.Visible:= false;
   UpDownBreitePaddle.Visible:= false;
   // Textfelder
   TextfeldGeschwindigkeit.Visible:= false;
   TextfeldBreitePaddle.Visible:= false;
   // Rahmen Einstellungen
   Rahmen.Visible:= false;
   // Timer starten
   Timer1.Enabled:= true;
   // Geschwindigkeit lesen und festlegen
   GeschwindigkeitX:= UpDownGeschwindigkeit.Position;
   GeschwindigkeitY:= UpDownGeschwindigkeit.Position;
   // Breite des Paddles
   Paddle.Width:= UpDownBreitePaddle.Position;
   SpielstandZeigen;
   GeschwindigkeitAnzeigen;
end;
```

Anzeige der Geschwindigkeit

```
procedure TForm1.GeschwindigkeitAnzeigen;
begin
   LabelAnzeigeGeschwindigkeit.Caption:= 'Geschwindigkeit: ' +
   IntToStr(GeschwindigkeitX);
end;
```

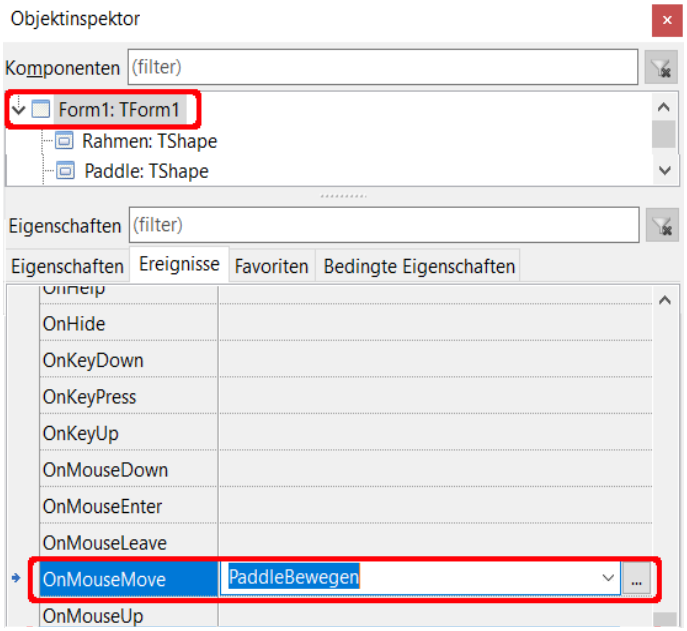
Spielstand anzeigen:

```
procedure TForm1.SpielstandZeigen;
begin
   // Anzahl Treffer anzeigen
   LabelSpielstand.Caption:= 'Treffer: ' + IntToStr(Treffer);
   // nach jeweils 10 Treffern → Geschwindigkeit erhöhen
   if Treffer > 0 then
   begin
      // wenn Anzahl Treffer ohne Rest (mod) durch 10 teilbar
     if Treffer mod 10 = 0 then
     begin
         // negative GeschwindigkeitX → GeschwindigkeitX verringern
         if GeschwindigkeitX < 0 then dec(GeschwindigkeitX)
          // positive Geschwindigkeit →  GeschwindigkeitX erhöhen
         else inc(GeschwindigkeitX);
          // GeschwindigkeitY ist in der Abwärtsbewegung immer positiv
         inc(GeschwindigkeitY);
         if GeschwindigkeitX < 0 then
           LabelAnzeigeGeschwindigkeit.Caption:= 'Geschwindigkeit: ' +
           IntToStr(-GeschwindigkeitX)
           else LabelAnzeigeGeschwindigkeit.Caption:= 'Geschwindigkeit: ' +
           IntToStr(GeschwindigkeitX);
       end;
   end;
end;
```

12 Erstelle in den Ereignissen der Form die Prozedure FormCreate. Sie sorgt dafür, dass beim Start des Programms die aktuellen Werte für die Geschwindigkeit und die Breite des Paddles eingetragen werden.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   TextfeldGeschwindigkeit.Text:= IntToStr(UpDownGeschwindigkeit.Position);
   TextfeldBreitePaddle.Text:= IntToStr(UpDownBreitePaddle.Position);
   LabelSpielEnde.Visible:= false;
   LabelAnzeigeGeschwindigkeit.Caption:= 'Geschwindigkeit: ' +
   IntToStr(UpDownGeschwindigkeit.Position);
end;
```

13 Jetzt kannst du mit der Programmierung des Paddles beginnen:




Setze als Name der Prozedur „PaddleBewegen“ und klicke anschließend auf die beiden Punkte.

Jetzt bewegt sich das Paddle, wenn die Maus bewegt wird. Allerdings ist die linke Kante des Paddles der Bezugspunkt für die Bewegung, es soll aber die Mitte des Paddles sein:

```
procedure TForm1.PaddleBewegen(Sender: TObject; Shift: TShiftState; X,
 Y: Integer);
begin
   // X →  Mausposition X-Achse
   // davon wird die Hälfte der Breite des Paddles abgezogen
   Paddle.Left:= X - Paddle.Width div 2;
   // das Paddle wird immer am unteren Ende platziert
   // auch wenn die Form in der Höhe verändert wird
   Paddle.Top:= Form1.Height - Paddle.Height;
end;
```

Für die Bewegung des Balls gibt es verschiedene Szenearien, die berücksichtigt werden müssen:

- der Ball stößt an die linke Begrenzung (Ball.Left<= 0)
- der Ball erreicht die rechte Begrenzung (Ball.Left + Ball.Width>= Form1.Width)
- der Ball berührt die Form oben (Ball.Top<= 0)
- der Ball prallt am Paddle ab
  linker Rand des Paddles → Ball.Left + Ball.Width>= Paddle.Left
  rechter Rand des Paddles → Ball.Left<= Paddle.Left + Paddle.Width
  oberer Rand des Paddles → Ball.Top + Ball.Height>= Paddle.Top
- der Ball erreicht das untere Ende der Form
  (Ball.Top + Ball.Height >= Form1.Height)

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   Ball.Left:= Ball.Left + GeschwindigkeitX;
   Ball.Top:= Ball.Top + GeschwindigkeitY;
   // Game Over -> Ball am unteren Ende
   if Ball.Top + Ball.Height >= Form1.Height then EndeSpiel;
   // rechts und links
   if (Ball.Left<= 0) or (Ball.Left + Ball.Width>= Form1.Width) then
   GeschwindigkeitX:=- GeschwindigkeitX;
   // oben -> Geschwindigkeit rückwärts
   if (Ball.Top<= 0) then
   begin
      GeschwindigkeitY:=- GeschwindigkeitY;
      inc(Treffer);
      SpielstandZeigen;
   end;
   // Ball berührt Paddle
   // linker Rand -> Ball.Left + Ball.Width>= Paddle.Left
   // rechter Rand -> Ball.Left<= Paddle.Left + Paddle.Width
   // oberer Rand -> Ball.Top + Ball.Height>= Paddle.Top
   if (Ball.Left + Ball.Width>= Paddle.Left) and (Ball.Left<= Paddle.Left
  + Paddle.Width) and (Ball.Top + Ball.Height>= Paddle.Top) then
   begin
      // Paddle getroffen -> Geschwindigkeit rückwärts
      GeschwindigkeitY:= -GeschwindigkeitY;
   end;
end;
```

14 Jetzt fehlen noch die Prozeduren zum Starten, Beenden und die Anzeigen für die Geschwindigkeit und „Game Over“ beim Überschreiten der unteren Begrenzung.

```
procedure TForm1.ButtonStartClick(Sender: TObject);
begin
   SpielStart;
end;

procedure TForm1.ButtonStoppSpielClick(Sender: TObject);
begin
   Application.Terminate;
end;

procedure TForm1.EndeSpiel;
begin
   LabelSpielEnde.Visible:= true;
   Timer1.Enabled:= false;
end;

procedure TForm1.GeschwindigkeitAnzeigen;
begin
   LabelAnzeigeGeschwindigkeit.Caption:= 'Geschwindigkeit: ' +
   IntToStr(GeschwindigkeitX);
end;
```

15 Das Einstellungsfenster soll bei einem Klick auf das Bild Einstellungen sichtbar werden. Das Bild findest du hier.

```
procedure TForm1.BildEinstellungenClick(Sender: TObject);
begin
   // Einstellungen anzeigen
   ButtonStart.Visible:= true;
   LabelEinstellungen.Visible:= true;
   UpDownGeschwindigkeit.Visible:= true;
   UpDownBreitePaddle.Visible:= true;
   Rahmen.Visible:= true;
   LabelGeschwindigkeit.Visible:= true;
   LabelBreitePaddel.Visible:= true;
   TextfeldGeschwindigkeit.Visible:= true;
   TextfeldBreitePaddle.Visible:= true;
   ButtonStoppSpiel.Visible:= true;
   // Ball sichtbar platzieren
   Ball.Top:= ClientWidth div 2;
   Ball.Left:= ClientHeight div 2;
   // aktuelle Geschwindigkeit anzeigen
   // prüfen ob Geschwindigkeit negativ ist
   if GeschwindigkeitX < 0 then GeschwindigkeitX:= -GeschwindigkeitX;
   TextfeldGeschwindigkeit.Text:= IntToStr(GeschwindigkeitX);
   UpDownGeschwindigkeit.Position:= GeschwindigkeitX;
   // Timer stoppen
   Timer1.Enabled:= false;
end;
```

16 Du kannst das Programm jetzt starten.


Durch einen Klick in die Form soll das Programm pausieren und nach einem weiteren Klick wieder starten:

```
procedure TForm1.FormClick(Sender: TObject);
begin
   // Spiel pausieren
   // wenn Timer gestartet → Timer stoppen
   // wenn Timer gestoppt → Timer starten
   if Timer1.Enabled = true then Timer1.Enabled:= false
   else Timer1.Enabled:= true;
end;
```

Das Mausrad nach unten verringert die Geschwindigkeit:

```
procedure TForm1.FormMouseWheelDown(Sender: TObject; Shift: TShiftState;
  MousePos: TPoint; var Handled: Boolean);
var
  WertX: Integer;
begin
  // verhindert Werte < 0 → WertX wird immer positiv
  if GeschwindigkeitX > 0 then WertX:= GeschwindigkeitX
  else WertX:= -GeschwindigkeitX;

  if WertX > 1 then
  begin
     // von links nach unten
     if (GeschwindigkeitX > 0) and (GeschwindigkeitY > 0) then
     begin
       dec(GeschwindigkeitX);
       dec(GeschwindigkeitY);
       GeschwindigkeitAnzeigen;
     end;

     // von links nach oben
     if (GeschwindigkeitX > 0) and (GeschwindigkeitY < 0) then
     begin
       GeschwindigkeitY:= -GeschwindigkeitY;
       dec(GeschwindigkeitX);
       dec(GeschwindigkeitY);
       GeschwindigkeitAnzeigen;
       GeschwindigkeitY:= -GeschwindigkeitY;
     end;

     // von rechts nach unten
     if (GeschwindigkeitX < 0) and (GeschwindigkeitY > 0) then
     begin
       GeschwindigkeitX:= -GeschwindigkeitX;
       dec(GeschwindigkeitX);
       dec(GeschwindigkeitY);
       GeschwindigkeitAnzeigen;
       GeschwindigkeitX:= -GeschwindigkeitX;
    end;

    // von rechts nach oben
    if (GeschwindigkeitX < 0) and (GeschwindigkeitY < 0) then
    begin
       GeschwindigkeitX:= -GeschwindigkeitX;
       GeschwindigkeitY:= -GeschwindigkeitY;
       dec(GeschwindigkeitX);
       dec(GeschwindigkeitY);
       GeschwindigkeitAnzeigen;
       GeschwindigkeitX:= -GeschwindigkeitX;
       GeschwindigkeitY:=-GeschwindigkeitY;
     end;
  end;
end;
```

Das Mausrad nach oben vergrößert die Geschwindigkeit:

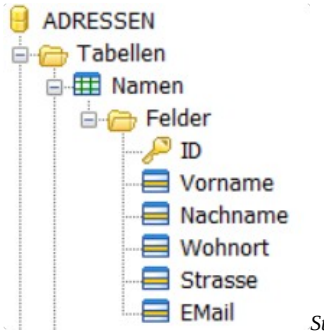
```
procedure TForm1.FormMouseWheelUp(Sender: TObject; Shift: TShiftState;
  MousePos: TPoint; var Handled: Boolean);
begin
   // von links nach unten
   if (GeschwindigkeitX > 0) and (GeschwindigkeitY > 0) then
   begin
     inc(GeschwindigkeitX);
     inc(GeschwindigkeitY);
     GeschwindigkeitAnzeigen;
   end;

   // von links nach oben
   if (GeschwindigkeitX > 0) and (GeschwindigkeitY < 0) then
   begin
     GeschwindigkeitY:= -GeschwindigkeitY;
     inc(GeschwindigkeitX);
     inc(GeschwindigkeitY);
     GeschwindigkeitAnzeigen;
     GeschwindigkeitY:= -GeschwindigkeitY;
   end;

   // von rechts nach unten
   if (GeschwindigkeitX < 0) and (GeschwindigkeitY > 0) then
   begin
     GeschwindigkeitX:= -GeschwindigkeitX;
     inc(GeschwindigkeitX);
     inc(GeschwindigkeitY);
     GeschwindigkeitAnzeigen;
     GeschwindigkeitX:= -GeschwindigkeitX;
   end;

   // von rechts nach oben
   if (GeschwindigkeitX < 0) and (GeschwindigkeitY < 0) then
   begin
     GeschwindigkeitX:= -GeschwindigkeitX;
     GeschwindigkeitY:= -GeschwindigkeitY;
     inc(GeschwindigkeitX);
     inc(GeschwindigkeitY);
     GeschwindigkeitAnzeigen;
     GeschwindigkeitX:= -GeschwindigkeitX;
     GeschwindigkeitY:=-GeschwindigkeitY;
   end;
end;
```

Hier kannst du dir eine erweiterte Pong-Version ansehen und das Programm herunterladen.

# Datenbank mit SQL

Hier kannst du die benötigten Dateien herunterladen:
Adressen.db, sqlite3.dll, SQL-Datenbank.exe
Das Beispielprogramm wird etwas modifiziert beschrieben.

Wenn du mit der 64-Bit Version von Lazarus arbeitest, musst du diese Datei herunterladen und sie in sqlite3.dll umbenennen.




*Struktur der Datenbank Adressen.db*

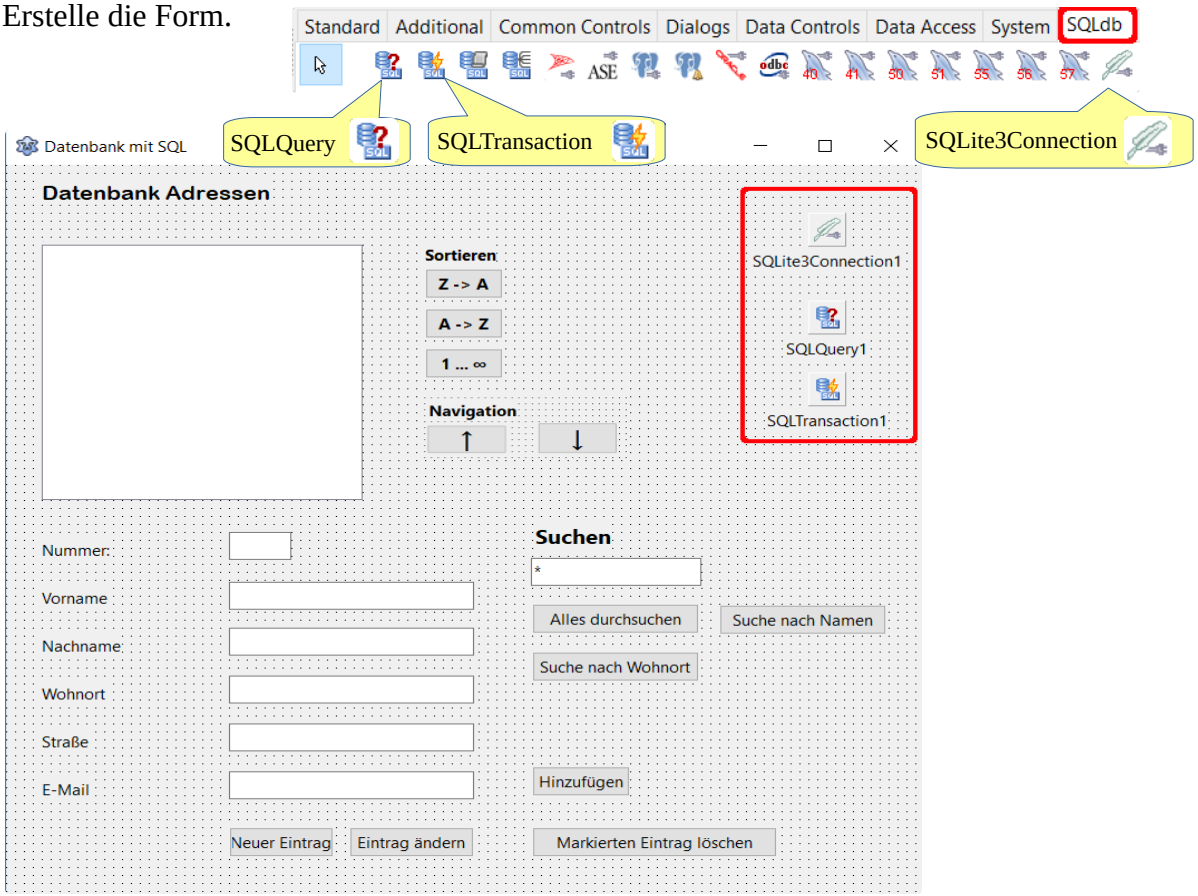
SQL ist eine Datenbanksprache. Die Daten werden in Tabellen und Feldern abgelegt.

Ein Programm zur Administrieren und Editieren von SQL-Datenbanken findest du hier.

| ID | Vorname | Nachname | Wohnort | Strasse | EMail |
|---|---|---|---|---|---|
| 1 | Karl | Winter | Koblenz | Hafenstr. 12 | karl@winter.de |
| 2 | Monika | Unger | Trier | Moselstr. 15 | monika.unger.de |
| 3 | Udo | Zumdick | Rostock | Badstr. 2 | udo@zumdick.de |
| 4 | Klara | Laufenberg | Stralsund | Ostseechaussee 1023 | klara@laufenberg.de |

*Inhalt der Datenbank Adressen.db (Auszug)*

1 Erstelle die Form.



2 Definiere die Variablen:

```
var
  Form1: TForm1;
  // dynamische Arrays für die Edit-Felder
  AnzeigeNummer: Array of String;
  AnzeigeVorname: Array of String;
  AnzeigeNachname: Array of String;
  AnzeigeWohnort: Array of String;
  AnzeigeStrasse: Array of String;
  AnzeigeEMail: Array of String;
  // das aktuelle Element
  Zaehler: Integer;
  // maximale Anzahl Datensätze
  DatenMax: Integer;
  // alphabetische Sortierung
  // true → abwärts, false → aufwärts
  AlphaSort : Boolean;
  // sortieren nach Nummer → true
  IDSort: Boolean;
  // wenn Liste neu geladen wird (true) → ersten Eintrag markieren
  ListeNeuLaden: Boolean;
```

3 Die Prozedur DatenZeigen:

Du musst den Namen der Prozedur selbst in die Liste der Prozeduren im Kopf des Programms eintragen!

```
procedure TForm1.DatenZeigen;
begin
   Zaehler:= 0;
   // ListBox leeren
   ListBox1.Clear;
   // Datenbankabfrage schließen
   SQLQuery1.Close;
   // Sortieren nach Nummern
   If IDSort = true then
   begin
      // Abfrage der Daten:
      // SELECT → Auswahl, * → alle Daten, FROM Namen → Tabelle Namen
      // ORDER BY ID → nach Feld ID sortieren
      SQLQuery1.SQL.text:='SELECT * FROM Namen ORDER BY ID';
   end;
   // asc aufwärts desc abwärts sortieren
   if IDSort = false then
   begin
      if AlphaSort = true then
      begin
         SQLQuery1.SQL.text:='SELECT * FROM Namen ORDER BY Nachname asc'
      end
      else SQLQuery1.SQL.text:='SELECT * FROM Namen ORDER BY Nachname desc';
   end;
```

```
    SQLQuery1.Open;
    // solange das Dateiende (Eof → End of file) nicht erreicht ist
    while not SQLQuery1.Eof do
    begin
       // Arrays um 1 vergrößern
       SetLength(AnzeigeNummer, Zaehler + 1);
       SetLength(AnzeigeVorname, Zaehler + 1);
       SetLength(AnzeigeNachname, Zaehler + 1);
       SetLength(AnzeigeWohnort, Zaehler + 1);
       SetLength(AnzeigeStrasse, Zaehler + 1);
       SetLength(AnzeigeEMail, Zaehler + 1);
       // die Felder der Tabelle den Edit-Feldern zuordnen
       AnzeigeNummer[Zaehler]:= SQLQuery1.Fields[0].AsString;
       AnzeigeVorname[Zaehler]:= SQLQuery1.Fields[1].AsString;
       AnzeigeNachname[Zaehler]:= SQLQuery1.Fields[2].AsString;
       AnzeigeWohnort[Zaehler]:= SQLQuery1.Fields[3].AsString;
       AnzeigeStrasse[Zaehler]:= SQLQuery1.Fields[4].AsString;
       AnzeigeEMail[Zaehler]:= SQLQuery1.Fields[5].AsString;
       // Anzeige Nummer Fields[0], Nachname Field[2]
       // Vorname Fields[1]
       ListBox1.Items.add(SQLQuery1.Fields[0].AsString +' ' +
       SQLQuery1.Fields[2].AsString + ', ' + SQLQuery1.Fields[1].AsString);
       // nächste Zeile
       SQLQuery1.Next;
       // Zaehler erhöhen
       inc(Zaehler);
  end;
  // Nummer des letzten Datensatzes sichern
  // wird beim Anlegen eines neuen Datensatzes verwendet
  DatenMax:= Zaehler;
  AlphaSort:= true;
  IDSort:= false;
  // die Liste wurde geändert → neu laden
  If ListeNeuLaden = true then
  begin
     // ersten Eintrag markieren
     ListBox1.Selected[0]:= true;
     // erneutes Neuladen verhindern
     ListeNeuLaden := false;
     // Edit-Felder mit den Daten füllen
     EditNummer.Text:= AnzeigeNummer[ListBox1.ItemIndex];
     EditVorname.Text:= AnzeigeVorname[ListBox1.ItemIndex];
     EditNachname.Text:= AnzeigeNachname[ListBox1.ItemIndex];
     EditOrt.Text:= AnzeigeWohnort[ListBox1.ItemIndex];
     EditStrasse.Text:= AnzeigeStrasse[ListBox1.ItemIndex];
     EditMail.Text:= AnzeigeEMail[ListBox1.ItemIndex];
   end;
end;
```

4 Die Prozedur `FormCreate` bestimmt die Datenbank und stellt die Verbindung her:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   try
      // Name der Datenbank
      SQLite3Connection1.DatabaseName:='Adressen.db';
      // Verbindung mit der Datenbank vorbereiten
      SQLTransaction1.Database:= SQLite3Connection1;
      SQLQuery1.Transaction:= SQLTransaction1;
      // Verbindung testen -> keine Verbindung -> weiter mit Except
      SQLQuery1.Open;
      SQLQuery1.Close;
   except
      begin
         ShowMessage('Die Datenbank wurde nicht gefunden' + sLineBreak +
         'Das Programm wird beendet!');
         Application.Terminate;
      end;
   end;
   // die Liste wird nach Nummern (ID) sortiert
   AlphaSort:= false;
   IDSort:= true;
   // beim Start/Ändern der Liste wird der erste Eintrag der Liste
   // markiert
   ListeNeuLaden:= true;
end;
```

5 Nach unten navigieren:

```
procedure TForm1.ButtonVorClick(Sender: TObject);
begin
   // wenn der aktuelle Eintrag < Ende der Liste
   if ListBox1.ItemIndex < ListBox1.Count - 1 then
   begin
      // Edit-Felder aktualisieren
      EditNummer.Text:= AnzeigeNummer[ListBox1.ItemIndex + 1];
      EditVorname.Text:= AnzeigeVorname[ListBox1.ItemIndex + 1];
      EditNachname.Text:= AnzeigeNachname[ListBox1.ItemIndex + 1];
      EditOrt.Text:= AnzeigeWohnort[ListBox1.ItemIndex + 1];
      EditStrasse.Text:= AnzeigeStrasse[ListBox1.ItemIndex + 1];
      EditMail.Text:= AnzeigeEMail[ListBox1.ItemIndex + 1];
      // aktuellen Eintrag markieren
      ListBox1.Selected[ListBox1.ItemIndex + 1]:= true;
   end;
end;
```

5 Nach oben navigieren:

```
procedure TForm1.ButtonZurueckClick(Sender: TObject);
begin
   // der aktuelle Eintrag muss > 0 sein
   if ListBox1.ItemIndex > 0 then
   begin
      // Edit-Felder aktualisieren
      EditNummer.Text:= AnzeigeNummer[ListBox1.ItemIndex - 1];
      EditVorname.Text:= AnzeigeVorname[ListBox1.ItemIndex - 1];
      EditNachname.Text:= AnzeigeNachname[ListBox1.ItemIndex - 1];
      EditOrt.Text:= AnzeigeWohnort[ListBox1.ItemIndex - 1];
      EditStrasse.Text:= AnzeigeStrasse[ListBox1.ItemIndex - 1];
      EditMail.Text:= AnzeigeEMail[ListBox1.ItemIndex - 1];
      EditDatumEintrag.Text:= AnzeigeDatumEintrag[ListBox1.ItemIndex];
      // aktuellen Eintrag markieren
      ListBox1.Selected[ListBox1.ItemIndex - 1]:= true;
   end;
end;
```

6 Datensatz löschen:

```
procedure TForm1.ButtonLoeschenClick(Sender: TObject);
begin
   SqlQuery1.Close;
   // Datenbankabfrage
   // DELETE → Datensatz löschen, ID ist der Inhalt des Edit-Feldes
   // EditNummer
   // da es ich um einen String handelt, muss der Wert in doppelte
   // Anführungszeichen gesetzt werden
   SqlQuery1.SQL.Text := 'DELETE FROM Namen
   WHERE ID = "'+ EditNummer.Text +'"';
   // Abfrage ausführen
   SqlQuery1.ExecSQL;
   // Daten dauerhaft speichern (Commit)
   SqlTransaction1.Commit;
   // Liste zeigen
   DatenZeigen;
end;
```

7 Eintrag ändern:

```
procedure TForm1.ButtonAendernClick(Sender: Tobject);
// lokale Variablen definieren, werden nur in der Prozedur verwendet
var
    Nachname, Vorname, Wohnort, ID, EMail, Strasse: String;
begin
   // Inhalt der Edit-Felder den Variablen zuordnen
   Vorname := EditVorname.Text;
   Nachname := EditNachname.Text;
   Wohnort:= EditOrt.Text;
   Strasse := EditStrasse.Text;
   EMail := EditMail.Text;
   ID:= EditNummer.Text;
   SqlQuery1.Close;
   // UPDATE → Datensatz aktualisieren
   // SET Feld schreiben
   // Format: UPDATE SET Datenbankfeld1 = Wert1, Datenbankfeld2 = Wert2
   // WHERE ID = IDWert
   // Nachname = + "'+ Nachname + '" → Feld Nachname wird mit dem Inhalt
   // des Strings Nachname überschrieben
   // die Variable muss in doppelte Anführungszeichen gesetzt werden
   SqlQuery1.Sql.Text:= 'UPDATE Namen SET Nachname = + "'+ Nachname + '" ,
   Vorname = + "'+ Vorname + '" , Wohnort = + "'+ Wohnort +'" ,
   Strasse = + "'+ Strasse + '" , EMail = + "' + EMail + '"
   WHERE ID = "'+ ID +'"';
   // Abfrage ausführen
   SqlQuery1.ExecSQL;
   // Daten dauerhaft schreiben (Commit)
   SQLTransaction1.Commit;
   // Edit-Felder leeren
   EditVorname.Text:= '';
   EditNachname.Text:= '';
   EditNummer.Text:= '';
   EditOrt.Text:= '';
   EditStrasse.Text:= '';
   EditMail.Text:= '';
   EditDatumEintrag.Text:= '';
   // Liste wurde verändert → wird neu geladen
   ListeNeuLaden:= true;
   // Liste zeigen
   DatenZeigen;
end;
```

8 Datensatz hinzufügen:

```
procedure TForm1.ButtonNeuClick(Sender: TObject);
begin
   // Button Hinzufügen sichtbar machen
   ButtonHinzu.Visible:= true;
   // Edit-Felder leeren
   EditVorname.Text:= '';
   EditNachname.Text:= '';
   EditNummer.Text:= '';
   EditOrt.Text:= '';
   EditStrasse.Text:= '';
   EditMail.Text:= '';
   // Nummer des neuen Datensatzes
   EditNummer.Text:= IntToStr(DatenMax + 1);
end;
```

9 Ein Klick auf den Button Hinzufügen schreibt den Datensatz:

```
procedure TForm1.ButtonHinzuClick(Sender: Tobject);
// lokale Variablen definieren, werden nur in der Prozedur verwendet
var
  Nachname, Vorname, Wohnort, ID, Strasse, EMail: String;
begin
   // Inhalt der Edit-Felder den Variablen zuordnen
   Nachname := EditVorname.text;
   Vorname := EditNachname.text;
   ID:= EditNummer.Text;
   Wohnort:= EditOrt.Text;
   Strasse:= EditStrasse.Text;
   EMail:= EditMail.Text;
   SqlQuery1.Close;
   SQLQuery1.SQL.Text:='INSERT INTO Namen VALUES ("'+ ID + '","' +
   Nachname + '","' + Vorname + '","' + Wohnort  + '","' + Strasse  +
   '","' + EMail + '")';
   // Abfrage ausführen
   SqlQuery1.ExecSQL;
   // Daten dauerhaft speichern (Commit)
   SQLTransaction1.Commit;
   // Edit-Felder leeren
   EditVorname.Text:= '';
   EditNachname.Text:= '';
   EditNummer.Text:= '';
   EditOrt.Text:= '';
   EditStrasse.Text:= '';
   EditMail.Text:= '';
   // Liste wurde verändert → wird neu geladen
   ListeNeuLaden:= true;
   // Liste zeigen
   DatenZeigen;
   // Button Hinzufügen wieder ausblenden
   ButtonHinzu.Visible:= false;
end;
```

10 Das Sortieren der Daten aufsteigend, absteigend oder nummerisch wird durch das Setzen der Variablen AlphaSort und IDSort in Gang gesetzt:

AlphaSort false: alphabetisch aufsteigend sortieren
AlphaSort true: alphabetisch absteigend sortieren
IDSort true: nach Nummer sortieren

```
procedure TForm1.ButtonZAClick(Sender: TObject);
begin
   AlphaSort:= false;
   ListeNeuLaden:= true;
   DatenZeigen;
end;

procedure TForm1.ButtonAZClick(Sender: TObject);
begin
   AlphaSort:= true;
   ListeNeuLaden:= true;
   DatenZeigen;
end;

procedure TForm1.ButtonNummerClick(Sender: TObject);
begin
   IDSort:= true;
   ListeNeuLaden:= true;
   DatenZeigen;
end;
```

11 Erstelle eine Prozedur DatenLaden. Sie wird bei den Suchfunktionen aufgerufen.


Du musst den Namen der Prozedur selbst in die Liste der Prozeduren im Kopf des Programms eintragen!

```
procedure TForm1.DatenLaden;
begin
   while not SQLQuery1.Eof do
   begin
      // Arrays vergrößern
      SetLength(AnzeigeNummer, Zaehler + 1);
      SetLength(AnzeigeVorname, Zaehler + 1);
      SetLength(AnzeigeNachname, Zaehler + 1);
      SetLength(AnzeigeWohnort, Zaehler + 1);
      SetLength(AnzeigeStrasse, Zaehler + 1);
      SetLength(AnzeigeEMail, Zaehler + 1);
      AnzeigeNummer[Zaehler]:= SQLQuery1.Fields[0].AsString;
      AnzeigeVorname[Zaehler]:= SQLQuery1.Fields[1].AsString;
      AnzeigeNachname[Zaehler]:= SQLQuery1.Fields[2].AsString;
      AnzeigeWohnort[Zaehler]:= SQLQuery1.Fields[3].AsString;
      AnzeigeStrasse[Zaehler]:= SQLQuery1.Fields[4].AsString;
      AnzeigeEMail[Zaehler]:= SQLQuery1.Fields[5].AsString;
      AnzeigeDatumEintrag[Zaehler]:= SQLQuery1.Fields[6].AsString;
      // Anzeige Nachname, Vorname
      ListBox1.Items.add(SQLQuery1.Fields[0].AsString +' ' +
      SQLQuery1.Fields[2].AsString + ', ' + SQLQuery1.Fields[1].AsString);
```

```
      SQLQuery1.Next;
      inc(Zaehler);
   end;
   // Daten gefunden
   if Zaehler > 0 then
   begin
      ListBox1.Selected[0]:= true;
      EditNummer.Text:= AnzeigeNummer[ListBox1.ItemIndex];
      EditVorname.Text:= AnzeigeVorname[ListBox1.ItemIndex];
      EditNachname.Text:= AnzeigeNachname[ListBox1.ItemIndex];
      EditOrt.Text:= AnzeigeWohnort[ListBox1.ItemIndex];
      EditStrasse.Text:= AnzeigeStrasse[ListBox1.ItemIndex];
      EditMail.Text:= AnzeigeEMail[ListBox1.ItemIndex];
      EditDatumEintrag.Text:= AnzeigeDatumEintrag[ListBox1.ItemIndex];
   end;
end;
```

### Alle Felder der Datenbank durchsuchen

```
procedure TForm1.ButtonSuchenAlleClick(Sender: TObject);
var
  Suchfeld: String;
begin
   Zaehler:= 0;
   ListBox1.Clear;
   Suchfeld:= EditSuchfeld.Text;
   // SQL verwendet als Standard das %-Zeichen als Platzhalter
   // für beliebige Zeichen, geläufiger ist aber das *-Zeichen
   // * durch % ersetzen
   // rfReplaceAll → alle Vorkommen ersetzen
   // rfIgnoreCase → Groß-/Kleinschreibung ignorieren
   Suchfeld:= StringReplace(EditSuchfeld.Text, '*', '%', [rfReplaceAll,
   rfIgnoreCase]);
   SQLQuery1.Close;
   // es werden alle Datensätze gesucht, in denen der Inhalt (WHERE) des
   // Suchfeldes entweder (OR) dem Nachnamen, Vorname … entspricht (LIKE)
   SQLQuery1.SQL.Text:= 'SELECT * FROM Namen WHERE Nachname LIKE  + "' +
   Suchfeld +'"   OR ID LIKE  + "'+ Suchfeld +'" OR Vorname LIKE  + "' +
   Suchfeld +'" OR Wohnort LIKE  + "'+ Suchfeld +'" OR Datum LIKE  + "' +
   Suchfeld +'"';
   SQLQuery1.Open;
   DatenLaden;
   DatenMax:= Zaehler;
end;
```

### Nachname suchen

```
procedure TForm1.ButtonSucheNameClick(Sender: TObject);
var
  Suchfeld: String;
begin
   Zaehler:= 0;
   ListBox1.Clear;
   Suchfeld:= EditSuchfeld.Text;
   // * durch % ersetzen
```

```
   Suchfeld:= StringReplace(EditSuchfeld.Text, '*', '%', [rfReplaceAll,
   rfIgnoreCase]);
   SQLQuery1.Close;
   SQLQuery1.SQL.Text:= 'SELECT * FROM Namen WHERE Nachname LIKE  + "' +
   Suchfeld +'"';
   SQLQuery1.Open;
   DatenLaden;
end;
```

**Wohnort suchen**

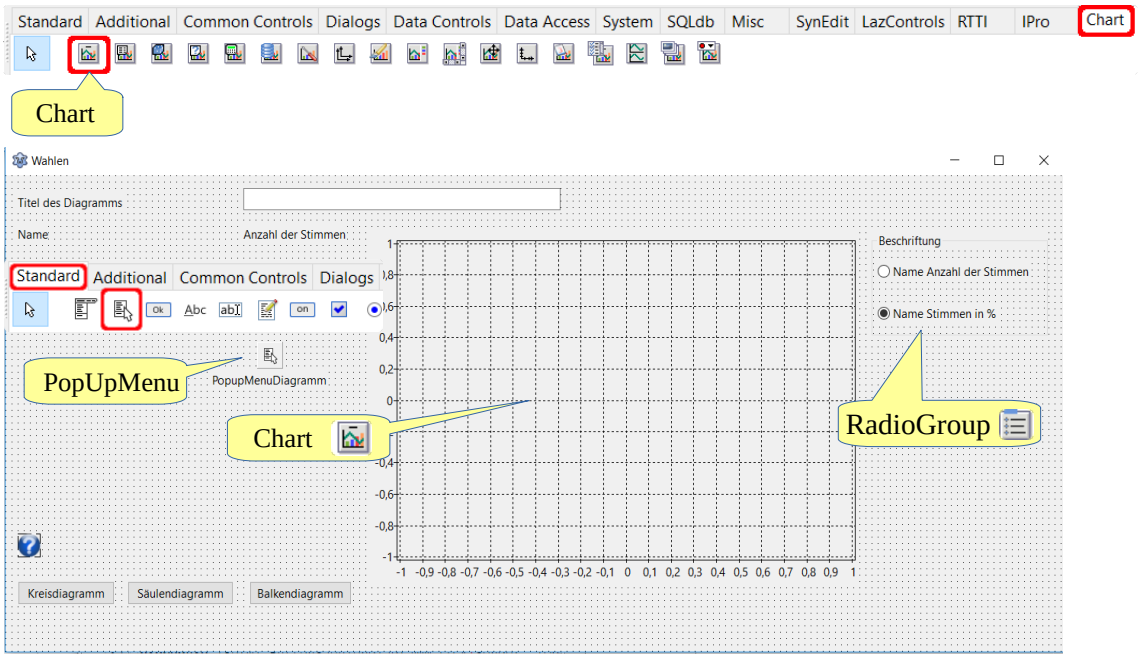
```
procedure TForm1.ButtonSucheOrtClick(Sender: TObject);
var
  Suchfeld: String;
begin
   Zaehler:= 0;
   ListBox1.Clear;
   Suchfeld:= EditSuchfeld.Text;
   // * durch % ersetzen
   Suchfeld:= StringReplace(EditSuchfeld.Text, '*', '%', [rfReplaceAll,
   rfIgnoreCase]);
   SQLQuery1.Close;
   // weil der Wohnort aus PLZ und Wohnort besteht, soll die Suche
   // erst nach der PLZ (ab Position 7 → SUBSTR) beginnen und
   // bis zum Ende des Strings LENGTH(Wohnort)durchgeführt werden
   // SUBSTR(String, StartPosition, EndPosition)
   SQLQuery1.SQL.Text:= 'SELECT * FROM Namen WHERE SUBSTR(Wohnort, 7,
   LENGTH(Wohnort)) LIKE  + "'+ Suchfeld +'"';
   SQLQuery1.Open;
   DatenLaden;
end;
```

12 Fertig, das Programm ist startklar.

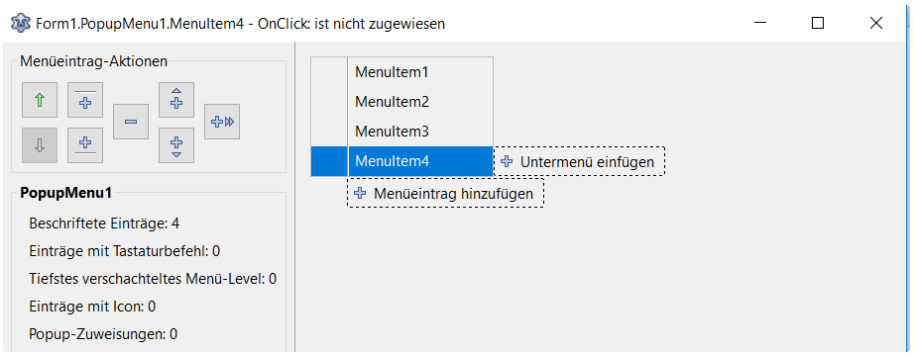
# Daten in Diagrammen darstellen

Mit Lazarus lassen sich verschiedene Diagramme darstellen.
Hier kannst du dir das Programm ansehen und das Beispielprogramm herunterladen.

1 Erstelle die Form.

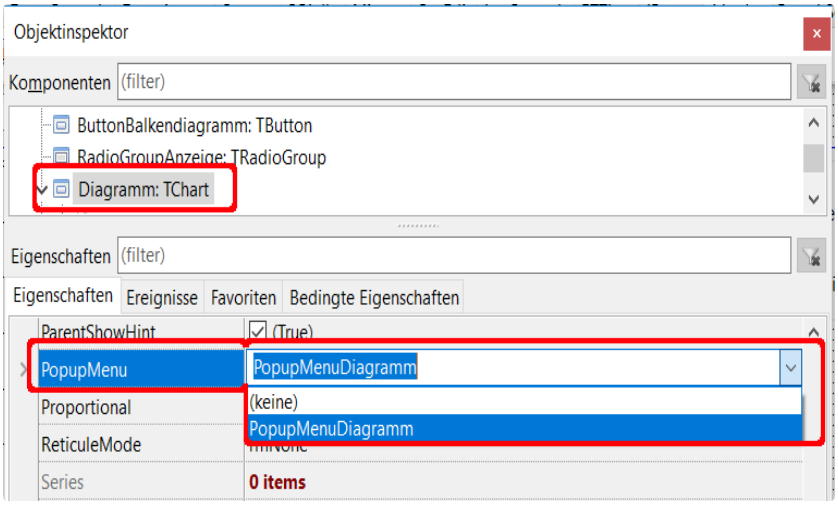



2 Klicke mit der rechten Maustaste auf das PopUpMenu und starte den Menüeditor.
Füge anschließend vier Einträge hinzu:

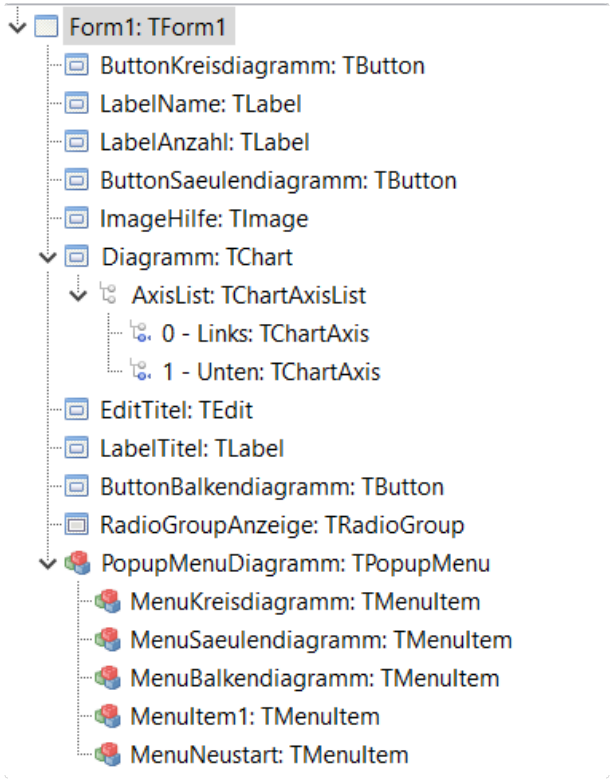



Verändere jeweils die Eigenschaft „Caption“ in „Kreisdiagramm“, „Säulendiagramm“ „Balkendiagramm“ und „Neustart“.

Setze die Eigenschaft PopUpMenu des Diagramms:




3 Liste der Elemente:



4 Unter `type` und unter `var` sind Ergänzungen erforderlich:

```
type

{ TForm1 }
TForm1 = class(TForm)
. . .
// TBarSerie -> BalkenDiagramm
Balken : TBarSeries;
// TBarSerie -> Säuleniagramm
Saeulen : TBarSeries;
// TPieSeries -> Kreisdiagramm
Kreisteile : TPieSeries;

var
  // Arrays für die Edit-Felder
  TextFeld: Array [1..6] of TEdit;
  ZahlFeld: Array [1..6] of TEdit;
  // AnzeigeLabel → Wert, der durch die Radiobuttons verändert wird
  // ButtonNummer → angeklickter Button (Kreis-/Säulen-/Balkendiagramm
  AnzeigeLabel, ButtonNummer: Integer;
```

5 In der Prozedur `FormCreate` werden die Edit-Felder erstellt:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Zaehler: Integer;
  AbstandOben, Zeilenabstand, AbstandLinks, Breite, Hoehe: Integer;
begin
   // Ausrichtung/Abstände der Edit-Felder in Pixeln
   AbstandOben:= 100;
   ZeilenAbstand:= 50;
   AbstandLinks:= 50;
   Breite:= 100;
   Hoehe:= 30;

   // Erstellen und Darstellung der Edit-Felder
   for Zaehler:= 1 to 6 do
   begin
      // Erstellen des Edit-Feldes Name
      TextFeld[Zaehler]:= TEdit.Create(self);
      // Zuordnung zu einer übergeordneten Komponente (hier die Form)
      TextFeld[Zaehler].Parent:= self;

      ZahlFeld[Zaehler]:=TEdit.Create(self);
      ZahlFeld[Zaehler].Parent:=self;

      TextFeld[Zaehler].Top:= AbstandOben;
      ZahlFeld[Zaehler].top:= AbstandOben;

      TextFeld[Zaehler].Left:= AbstandLinks;
      ZahlFeld[Zaehler].Left:= AbstandLinks + Breite * 3;
```

```
      TextFeld[Zaehler].Width:= Breite * 2;
      ZahlFeld[Zaehler].Width:= Breite div 2;

      TextFeld[Zaehler].Height:= Hoehe;
      ZahlFeld[Zaehler].Height:= Hoehe;

      // Feld wird rechts ausgerichtet
      ZahlFeld[Zaehler].Alignment:= taRightJustify;

      AbstandOben:= AbstandOben + ZeilenAbstand;
   end;

   // beim Überfahren des Fragezeichens wird der Hinweis gezeigt
   ImageHilfe.Hint:= 'Du musst mindestens zwei Datemsätze' + sLineBreak +
   'vollständig eingeben';
   ImageHilfe.ShowHint:= true;

   // Standard für die Anzeige der Werte im Diagramm als %
   AnzeigeLabel:= 2;
end;
```

## 6 Die Programmierung der Radiobuttons

```
procedure TForm1.RadioGroupAnzeigeClick(Sender: TObject);
begin
   // der Index bestimmt die Form der Anzeige
   // 1 → Anzeige als Wert, 2 → Anzeige als %
   case RadioGroupAnzeige.ItemIndex of
   0: begin
        AnzeigeLabel:= 1;
        // anschließend wird das Diagramm neu geladen
        // ButtonNummer bestimmt das Diagramm
        // 1 → Kreis, 2 → Säulen, 3 → Balken
        case ButtonNummer of
           1: ButtonKreisdiagrammClick(Sender);
           2: ButtonSaeulendiagrammClick(Sender);
           3: ButtonBalkendiagrammClick(Sender);
        end;
      end;
  1: begin
        AnzeigeLabel:= 2;
        case ButtonNummer of
           1: ButtonKreisdiagrammClick(Sender);
           2: ButtonSaeulendiagrammClick(Sender);
           3: ButtonBalkendiagrammClick(Sender);
        end;
      end;
  end;
end;
```

7 Die Erstellung des Kreisdiagramms:

```
procedure TForm1.ButtonKreisdiagrammClick(Sender: TObject);
var
  Zaehler, DatenMax: Integer;
  // Farben den Kreisteilen/Balken zuordnen
  Farben: array[1..6] of TColor = (clRed, clYellow, clFuchsia, clGreen, clBlue,
   clAqua);
begin
   // Anzahl der eingegebenen Daten feststellen
   DatenMax:= 1;
   // Nummer des angeklickten Buttons 1 -> Kreis, 2 -> Säulen, 3 -> Balken
   ButtonNummer:= 1;
   // feststellen, wie viele Datensätze eingegeben wurden
   for Zaehler:= 1 to Length(TextFeld) do
   begin
      // wenn beide Textfelder einen Wert haben -> DatenMax um 1 erhöhen
      if (Textfeld[zaehler].Text <> '') and (Zahlfeld[zaehler].Text <> '')
      then inc(DatenMax);
   end;
   // Diagramm nur anzeigen wenn > 2 Datensätze angelegt wurden
   if DatenMax > 2 then
   begin
      Diagramm.Visible:= true;
       RadioGroupAnzeige.Visible:= true;
      // evtl. vorhandenes Diagramm löschen
      Diagramm.ClearSeries;
      // Kreisteile erstellen (definiert unter var)
      Kreisteile:= TPieSeries.Create(Diagramm);
      Diagramm.AddSeries(Kreisteile);
      // Achsen verbergen
      Diagramm.AxisVisible := false;
      // wenn Titel leer ist
      if EditTitel.Text = '' then EditTitel.Text:= 'Wahlen';
      // Titel anzeigen
      Diagramm.Title.Text.Strings[0] := EditTitel.Text;
      Diagramm.Title.Font.Size := 12;
      Diagramm.Title.Visible := true;
      // Anzeige als Wert
      if AnzeigeLabel = 1 then Kreisteile.Marks.Style :=
      TSeriesMarksStyle(smsLabelValue)
      // Anzeige als %
      else Kreisteile.Marks.Style := TSeriesMarksStyle(smsLabelPercent);
      try
         for Zaehler := 1 to DatenMax - 1  do
         begin
            // Kreisteil hinzufügen
            // Zahlfeld[Zaehler].Text) -> Größe
            // Textfeld[Zaehler].Text -> Beschriftung
            // Farben[Zaehler] -> Farbedes Kreisteils
            Kreisteile.AddPie(StrToInt(Zahlfeld[Zaehler].Text),
            Textfeld[Zaehler].Text, Farben[Zaehler]);
         end;
       except ShowMessage('Es fehlen Angaben!');
       end;
   end
   // es sind weniger als 2 Datensätze erstellt worden
   else ShowMessage('Du musst mindestens zwei Datensätze anlegen!');
end;
```

8 Die Erstellung des Säulendiagramms:

```
procedure TForm1.ButtonSaeulendiagrammClick(Sender: TObject);
var
  Zaehler, DatenMax: Integer;
  Farben: array[1..6] of TColor = (clRed, clYellow, clFuchsia, clGreen, clBlue,
  clAqua);
begin
   DatenMax:= 1;
   ButtonNummer:= 2;
   for Zaehler:= 1 to Length(TextFeld) do
   begin
      if (Textfeld[zaehler].Text <> '') and (Zahlfeld[zaehler].Text <> '') then
      inc(DatenMax);
   end;
   // Diagramm nur anzeigen wenn > 2 Datensätze angelegt wurden
   if DatenMax > 2 then
   begin
      Diagramm.Visible:= true;
      RadioGroupAnzeige.Visible:= true;
      // vorhandenes Diagramm löschen
      Diagramm.ClearSeries;
      // Säulendiagramm erstellen
      Saeulen:= TBarSeries.Create(Diagramm);
      Diagramm.AddSeries(Saeulen);
      Saeulen.AxisIndexX:= 1;
      Saeulen.AxisIndexY:= 0;
      Diagramm.AxisVisible := true;
      // linke Achse anzeigen/untere Achse verbergen
      Diagramm.AxisList.BottomAxis.Visible := false;
      Diagramm.AxisList.LeftAxis.Visible := true;
      Diagramm.AxisList.BottomAxis.Marks.Source := Saeulen.Source;
      // Anzeige als Wert oder Prozent
      if AnzeigeLabel = 1 then Saeulen.Marks.Style :=
       TSeriesMarksStyle(smsLabelValue)
       else Saeulen.Marks.Style := TSeriesMarksStyle(smsLabelPercent);
       if EditTitel.Text = '' then EditTitel.Text:= 'Wahlen';
       Diagramm.Title.Text.Strings[0] := EditTitel.Text;
       Diagramm.Title.Font.Size := 12;
       Diagramm.Title.Visible := true;
       try
          for Zaehler := 1 to DatenMax - 1 do
          begin
             // Säule hinzufügen: Zahler -> x-Achse,
             // Zahlfeld[Zaehler].Text -> y-Achse
             // Textfeld[Zaehler].Text -> Beschriftung der Säule
             // Farben[Zaehler] -> Farbe der Säule
             Saeulen.AddXY(Zaehler, StrToInt(Zahlfeld[Zaehler].Text),
             Textfeld[Zaehler].Text, Farben[Zaehler]);
          end;
          except ShowMessage('Es fehlen Angaben!');
          end;
    end
  else ShowMessage('Du musst mindestens zwei Datensätze anlegen!');
end;
```

9 Die Erstellung des Balkendiagramms:

```
procedure TForm1.ButtonBalkendiagrammClick(Sender: TObject);
var
 Zaehler, DatenMax: Integer;
 Farben: array[1..6] of TColor = (clRed, clYellow, clFuchsia, clGreen, clBlue,
 clAqua);
begin
   DatenMax:= 1;
   ButtonNummer:= 2;
   for Zaehler:= 1 to Length(TextFeld) do
   begin
      if (Textfeld[zaehler].Text <> '') and (Zahlfeld[zaehler].Text <> '') then
      inc(DatenMax);
   end;
   // Diagramm nur anzeigen wenn > 2 Datensätze angelegt wurden
   if DatenMax > 2 then
   begin
      RadioGroupAnzeige.Visible:= true;
      Diagramm.Visible:= true;
      ButtonNummer:= 3;
      // Balken erstellen (definiert unter var)
      Balken:= TBarSeries.Create(Diagramm);
      Diagramm.ClearSeries;
      Diagramm.AddSeries(Balken);
      // Achsen sichtbar machen
      Diagramm.AxisVisible := true;
      // nur untere Achse anzeigen
      Diagramm.AxisList.BottomAxis.Visible := true;
      Diagramm.AxisList.LeftAxis.Visible := false;
      // Achsen x und y tauschen
      Balken.AxisIndexX:= 0;
      Balken.AxisIndexY:= 1;
      // linke Achse als Quelle festlegen
      Diagramm.AxisList.LeftAxis.Marks.Source := Balken.Source;
      if EditTitel.Text = '' then EditTitel.Text:= 'Wahlen';
      Diagramm.Title.Text.Strings[0] := EditTitel.Text;
      Diagramm.Title.Font.Size := 12;
      Diagramm.Title.Visible := true;
      if AnzeigeLabel = 1 then Balken.Marks.Style :=
      TSeriesMarksStyle(smsLabelValue)
      else Balken.Marks.Style := TSeriesMarksStyle(smsLabelPercent);
      try
      for Zaehler := 1 to DatenMax - 1 do
      begin
         // Balken hinzufügen: Zahler -> x-Achse,
         // Zahlfeld[Zaehler].Text -> y-Achse
         // Textfeld[Zaehler].Text -> Beschriftung der Säule
         // Farben[Zaehler] -> Farbe der Säule
         Balken.AddXY(Zaehler, StrToInt(Zahlfeld[Zaehler].Text),
         Textfeld[Zaehler].Text, Farben[Zaehler]);
      end;
      except ShowMessage('Es fehlen Angaben!');
      end;
   end
   else ShowMessage('Du musst mindestens zwei Datensätze anlegen!');
end;
```

10 Fertig, du kannst das Programm starten.

# Quellennachweise und Nutzungsbedingungen

Idee und Layout: Hartmut Waller

Fotos: Hartmut Waller

Grafiken: https://openclipart.org/

Lazarus: https://www.lazarus-ide.org/

Es ist erlaubt eine private Kopie zu speichern. Es ist ausdrücklich erwünscht, dieses Dokument für unterrichtliche Zwecke zu verwenden.

Eine Veröffentlichung auf anderen Internetseiten bedarf der Genehmigung des Autors.

Das Herunterladen von Programmdateien geschieht auf eigenes Risiko.

Ich versichere, dass diese Dateien nach bestem Wissen auf mögliche Viren oder Trojaner untersucht wurden.

Ich freue mich über Anregungen, Kritik und auch Lob.