Leibniz Universität
IT Services

Leibniz
Universität
Hannover

# Python -
# Grafische Oberflächen mit Tkinter erstellen

# Grafische Benutzeroberflächen

- Graphical User Interface (GUI)
- Schnittstelle zwischen Mensch und Maschine.
- Steuerung der Eingabe und Ausgabe von Daten.
- Erstellung mit Hilfe der Bibliothek Tkinter erstellt werden.
- Code-Dateien mit der Dateiendung „.pyw“.

# Aussehen der Benutzeroberfläche

- Corporate Desgin des Auftraggebers.
- Styleguides.
- Barrierefreiheit.
- Erwartung des Benutzers durch vorhandene Software, Formulare und so weiter.

# Was wird in der GUI abgebildet?

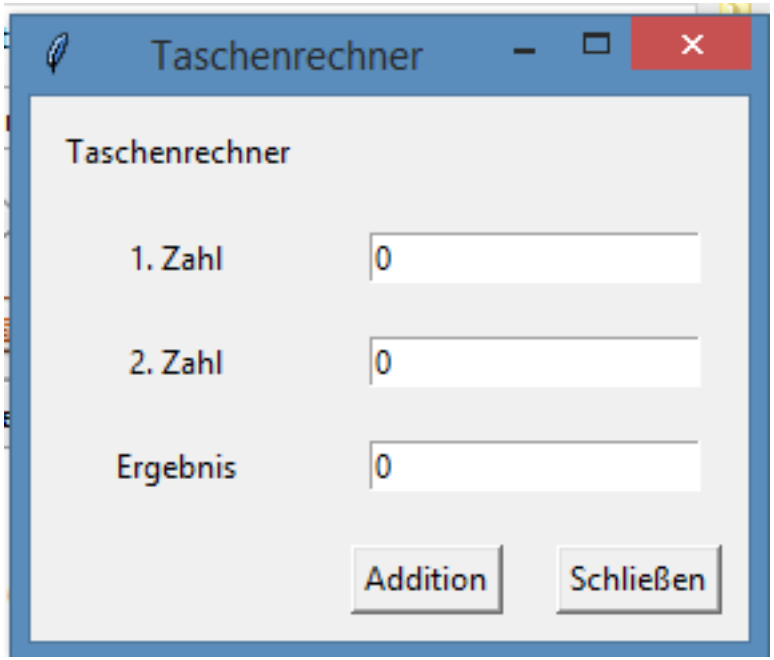
- Welche Arbeitsabläufe sollen abgebildet werden?
- In welcher Reihenfolge werden diese ausgeführt?
- Falls zu viele Felder abgebildet werden, kann die Aktion in verschiedene Aktivitäten unterteilt werden?

# Bestandteile einer GUI

- Fenster (window).
- Beliebige Anzahl von Steuerelementen (Widget).

# Fenster (window)

- Rechteckige Form.
- Begrenzung durch einen Rahmen
- Am oberen Rand eine Titelleiste.



# Widget

- Windows Gadget.
- Existiert nur in einem Fenster oder Container.
- Steuerelemente (Controls) und grafische Elemente zur Gestaltung.
- Interaktion mit dem Benutzer.

# Beispiele

- Textfelder ermöglichen die Eingabe von Text.
- Label (Beschriftungsfelder) zeigen Informationen zu Elementen im Fenster als Hilfe an.
- Auslösung von Aktionen (Ereignissen) mit Hilfe von Schaltflächen.
- Listenfelder, Kontrollkästchen sowie Optionsfelder ermöglichen eine Auswahl von Elementen.

# Auflistung von Widgets

- http://www.tkdocs.com/tutorial/widgets.html
- http://effbot.org/tkinterbook/tkinter-index.htm

# GUIs in Python

- Python bietet verschiedene Pakete zur Erstellung von GUIs.
- Informationen zu den Paketen finden Sie unter http://docs.python.org/py3k/faq/gui.html.

# Nutzung von Tkinter

- Gestaltung von grafischen Oberflächen.
- Standardbibliothek in Python.

# Informationen zu dem Paket

- http://docs.python.org/py3k/library/tk.html
- https://wiki.python.org/moin/TkInter
- https://www.python-kurs.eu/python_tkinter.php
- https://www.inf-schule.de/software/gui/entwicklung_tkinter
- http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

# Beispiel in Tkinter

```
from tkinter import *

root = Tk()
caption = "Herzlich Willkommen"
beschriftungsFeld = Label(root, text =  caption)
beschriftungsFeld.pack()
root.mainloop()
```

# Import der Bibliothek

```
from tkinter import *
```

- Von (from) dem Modul tkinter werden alle Elemente (*) importiert (import).

# Erzeugung des Fensters

```
root = Tk()
```

- Durch den Aufruf `Tk()` wird ein Fenster mit einer Titelleiste und einem Rahmen erzeugt.
- Ein Verweis auf dieses Fenster wird in der Variablen `root` gespeichert. Der Name ist frei wählbar.

# Beschriftungsfelder (Labels)

- Anzeige von statischen mehrzeiligen Text.
- Hilfe / Überschriften im Fenster.
- Häufig links oder oberhalb von anderen Widgets.

# Nutzung eines Bezeichnungsfeldes

```
beschriftungsFeld = Label(root, text =  caption)
```

- Mit Hilfe von Label() wird ein Bezeichnungsfeld erzeugt.
- Der erste Parameter der Initialisierungsmethode gibt Auskunft darüber, wo das Bezeichnungsfeld abgelegt wird.
- Der zweite Parameter text bekommt den anzuzeigenden Text übergeben.

# Einbettung des Widgets

```
beschriftungsFeld.pack()
```

- Mit Hilfe der Methode pack() wird das Widget in das Fenster eingebettet.

# Anzeigen des Fensters

```
root.mainloop()
```

- Das Label wird im Fenster aber erst angezeigt, wenn die Methode mainloop() aufgerufen wird.
- Ereignis-Schleife. Warten auf Aktionen des Benutzers.
- Sobald das Fenster geschlossen wird, wird die Schleife automatisch abgebrochen.

# Nutzung von Klassen

```
from tkinter import *

class Application(Frame):

    def __init__(self, master=None):
        pass

    def createWidgets(self):
        pass
```

# Klassenkopf

```
class Application(Frame):
```

- Jede Klassen-Definition beginnt mit dem Schlüsselwort `class`.
- Die Klasse hat einen eindeutigen Namen. Pro Datei wird eine Klasse erstellt.
- Die Klasse erbt von der Klasse `Frame`. Das Frame-Widget ist ein Rahmen. Der Container gruppiert Widgets wie Textfelder etc.

# … und aufrufen

```
root = Tk()
app = Application(master=root)
app.mainloop()
```

- Der Initialisierungsroutine der Klasse Application wird das Eltern-Fenster übergeben.
- Das Argument wird als Schlüssel-Wert-Paar übergeben.
- In diesem Beispiel wird das Fenster Tk() genutzt.

# Initialisierungsmethode

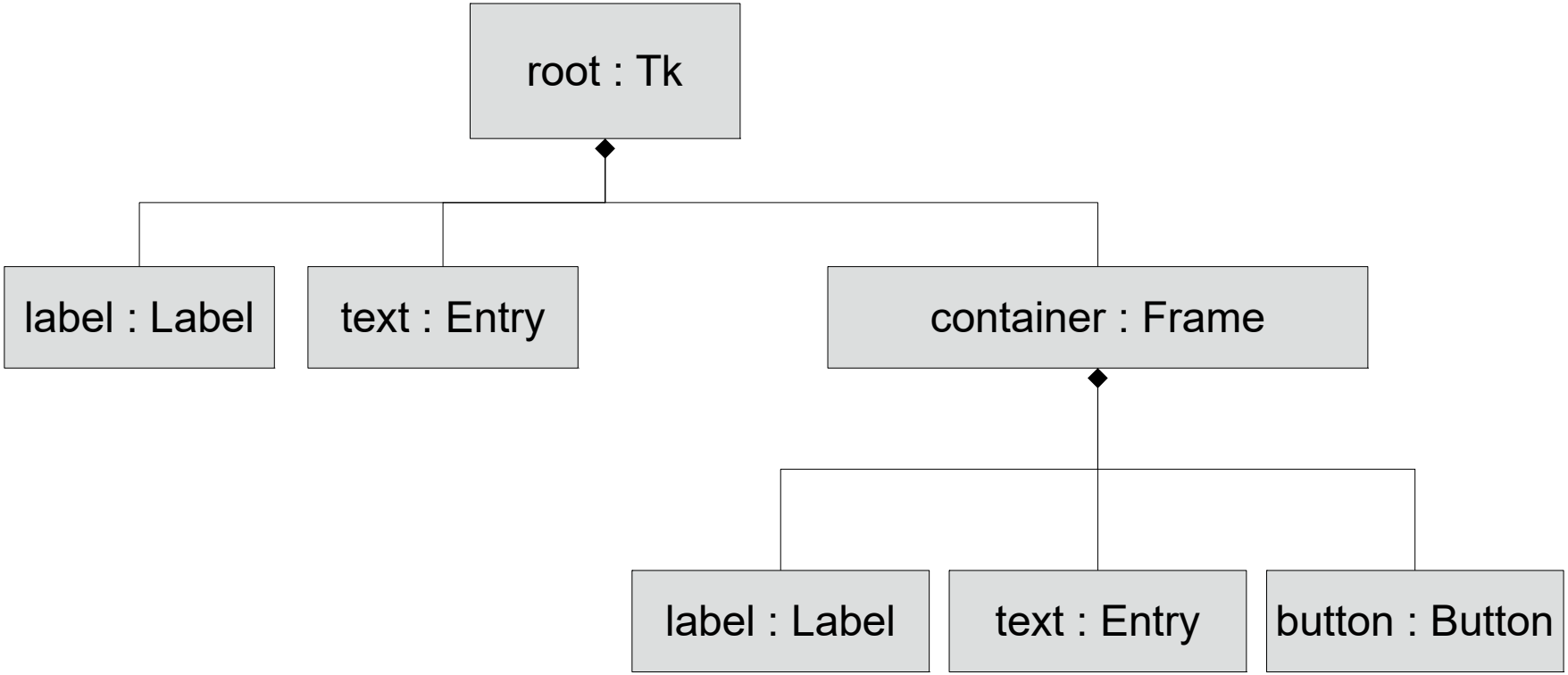
```
def __init__(self, master=None):
    Frame.__init__(self, master)
    self.pack()
    self.createWidgets()
```

- Zuerst wird ein Frame-Widget erzeugt. Die Initialisierungsmethode der Basisklasse wird aufgerufen.
- Anschließend wird der Container mit Hilfe der Methode `pack()` im Hauptfenster positioniert.
- Mit Hilfe der benutzerdefinierten Methode `createWidgets()` werden die Widgets im Container erzeugt und platziert.

# Container

- Behältnisse für andere Widgets.
- Gruppierung von Widgets, um eine Aktion abzubilden.
- Definition eines Layout für Benutzeroberflächen.
- Sobald das Eltern-Fenster geschlossen wird, werden alle darin enthaltenden Container geschlossen
- Die Größe des Containers wird durch den Inhalt festgelegt.

# Eltern-Kind (Master-Slave) - Beziehung



# Initialisierung der Basisklasse

```
Frame.__init__(self, master)
```

- Die Instanz sowie ein Verweis auf das Eltern-Fenster werden dem Konstruktor für ein Frame übergeben.
- Initialisierung der Basisklasse „Rechteckiger Container“.

# Erzeugung der Widgets

```
def createWidgets(self):
        ausgabeText = "Herzlich Willkommen"
        self.lblHallo = self.createLabel(ausgabeText)
        self.lblHallo.pack()
```

- Mit Hilfe dieser Methode werden alle Widgets erzeugt, die in dem rechteckigen Container abgelegt werden sollen.

# Erzeugung eines Beschriftungsfeldes

```
def createLabel(self, caption):
        beschriftungsFeld = Label(self,
                                  text=caption,
                                  bg='white', fg='#000000',
                                  font=('Helvetica', '18', 'bold'))

        return beschriftungsFeld
```

- Mit Hilfe von Label() wird ein Beschriftungsfeld im Container Frame erzeugt.
- Der Verweis auf das erzeugte Beschriftungsfeld wird an den Aufrufer zurückgegeben.

# Angezeigter Text

```
beschriftungsFeld = Label(self,
                          text=caption,
                          bg='white', fg='#000000',
                          font=('Helvetica', '18', 'bold'))
```

- Dem Schlüssel `text` wird der anzuzeigende Text übergeben.

# Schrift im Beschriftungsfeld

```
beschriftungsFeld = Label(self,
                          text=caption,
                          bg='white', fg='#000000',
                          font=('Helvetica', '18', 'bold'))
```

- Dem Schlüssel font wird ein Tupel von Strings übergeben.
- Das erste Element definiert die Schriftfamilie.
- Das zweite Element definiert die Schriftgröße.
- Das dritte Element definiert den Schriftschnitt.

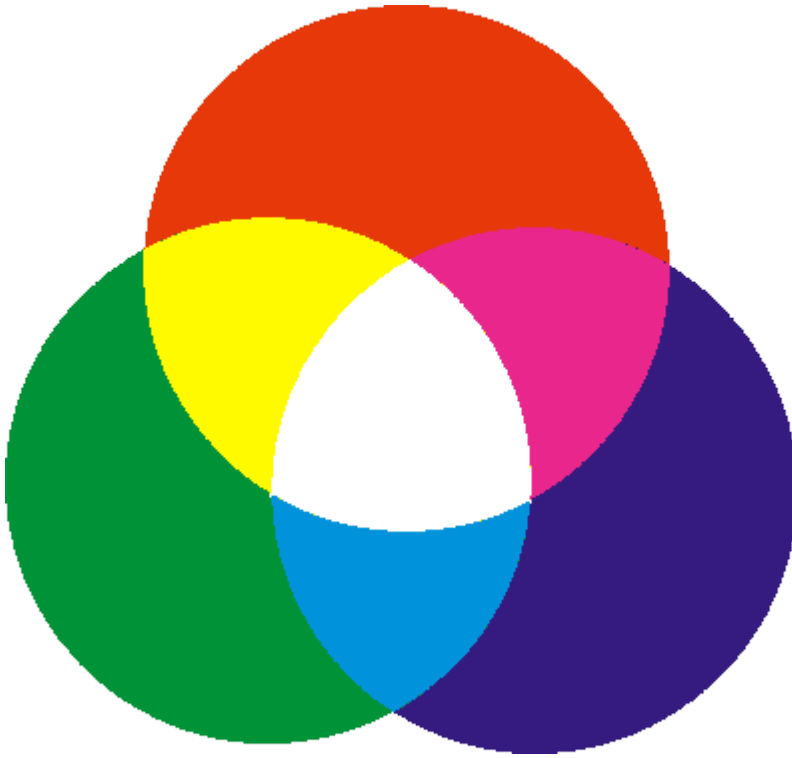
# Hinter- und Vordergrundfarbe

```
beschriftungsFeld = Label(self,
                          text=caption,
                          bg='white', fg='#000000',
                          font=('Helvetica', '18', 'bold'))
```

- Dem Schlüssel fg (foreground) wird eine Vordergrundfarbe als String oder Hexadezimalwert übergeben. Die Schriftfarbe wird definiert.
- Dem Schlüssel bg (background) wird eine Hintergrundfarbe als String oder Hexadezimalwert übergeben. Die Füllfarbe des Beschriftungsfeldes wird definiert.

# RGB-Farbsystem

- Red, green, blue.
- Darstellung von Farben am Bildschirm.
- Die Farben Rot, Grün, Blau werden 256 Helligkeitsstufen an.
- Die Farbe weiß hat eine Rot-Anteil von 255, einen Grün-Anteil von 255 und einen Blau-Anteil von 255.
- Um so mehr sich eine Farbe dem Wert 255 nährt, um so heller ist sie.



# Beispiele für Farben

| | | |
|---|---|---|
| | 'white' | '#FFFFFF' |
| | 'black' | '#000000' |
| | 'red' | '##FF0000' |
| | 'green' | '##00FF00' |
| | 'blue' | '##0000FF' |
| | 'yellow' | '##0000FF' |

# Hexadezimal-Werte

- Kennzeichnung mit dem Hash-Zeichen.
- Jeweils zwei Stellen jeden Farbanteil Rot, Grün und Blau.
- 00 stellt die Ganzzahl 0 dar. FF stellt die Zahl 255 dar.
- Für jede Stelle können die Zahlen 0 … 9 sowie die Buchstaben A … F genutzt werden. Die Buchstaben A … F symbolisieren die Ganzzahlen 10 bis 15,

# Informationen zu Farben

- https://wiki.tcl.tk/37701
- http://www.science.smith.edu/dftwiki/index.php/Color_Charts_for_Tkinter
- https://www.tutorialspoint.com/python/tk_colors.htm

# Erzeugung eines Textfeldes

```
def createTextfeld(self):
        textFeld = Entry(self)
        return textFeld
```

- In dieser Methode wird ein einzeiliges Textfeld `Entry()` erzeugt.
- Das Textfeld wird in dem Platzhalter `self` () positioniert. Das Widget wird in dem Frame dargestellt.
- Der Verweis auf das erzeugte Textfeld wird an den Aufrufer zurückgegeben.

# Einfügen von Text

```
textFeld.insert(0, default)
```

- Mit Hilfe der Methode .insert() wird Text in das Feld eingefügt.
- In diesem Beispiel wird an der Position 0 der String in der Variablen default eingefügt.

# Löschen von Text

```
textFeld.delete(0, END)
```

- Mit Hilfe der Methode .delete() wird Inhalt aus dem Textfeld entfernt.
- In diesem Beispiel wird alles von der Position 0 bis zum Ende des Text entfernt. Die Konstante END symbolisiert das letzte Zeichen in einem Textfeld. Das Textfeld wird vollständig geleert.

# Geometry Manager

- Layout-Manager.
- Wie werden Widgets und Container in einem Fenster angeordnet?
- Anzeige von Widgets und Container in einem Fenster.

# Nutzung von .pack()

```
self.pack()
self.lblBenutzername.pack(side=LEFT,pady=10,padx=10)
self.txtBenutzername.pack(fill=BOTH, expand=1)
```

- Die Anordnung der Container und Widgets erfolgt automatisiert.
- Die Widgets können mit Hilfe von Schlüssel-Wert-Paaren relativ zu ihrem Eltern-Container ausgerichtet werden.
- Nutzung für Fenster mit wenigen Widgets.

# … mit einer leeren Argumentliste

```
self.pack()
```

- Die Widgets werden aufeinander in der angegebenen Reihenfolge gestapelt.
- Die Gesamtheit aller Widgets füllt den Container vollständig aus.

# Größe des Widgets = Größe Container

```
self.txtBenutzername.pack(fill=BOTH, expand=1)
```

- Durch den Schlüssel fill kann das Widget in Abhängigkeit der Größe des Eltern-Containers horizontal (fill=X) und / oder senkrecht (fill=Y) expandieren.
- Falls der Schlüssel expand einen Wert größer als 1 hat, kann das Widget in Abhängigkeit der Größe des Eltern-Containers expandieren. Falls sich die Größe des Containers verändert, wird die Größe des Widgets automatisiert angepasst.

# Verankerung an eine Seite

```
self.lblBenutzername.pack(side=LEFT,pady=10,padx=10)
```

- Durch den Schlüssel side kann das Widget an einer bestimmten Seite des Eltern-Containers verankert werden. Möglichkeiten: CENTER, LEFT, RIGHT, TOP und BOTTOM.
- Mit Hilfe von pady und padx kann ein Abstand zwischen den Widgets festgelegt werden.

# Nutzung von .grid()

```
self.lblBenutzername.grid(row=0, padx=10, pady=10)
self.txtBenutzername.grid(column=1, row=1, padx=10, pady=10)
```

- Container und Widgets werden in Abhängigkeit von Zeilen und Spalten platziert.

# Angabe der Zeile

```
self.lblBenutzername.grid(row=0, padx=10, pady=10)
self.txtBenutzername.grid(column=1, row=1, padx=10, pady=10)
```

- Die erste Zeile (row) hat den Wert 0.
- Der Schlüssel rowspan verbindet Zeilen miteinander.

# Angabe der Spalte

```
self.lblBenutzername.grid(row=0, padx=10, pady=10)
self.txtBenutzername.grid(column=1, row=1, padx=10, pady=10)
```

- Die erste Spalte (column) in einer Zeile hat den Wert 0.
- Falls kein Wert angegeben wird, wird die erste freie Spalte in der angegebenen Zeile genutzt.
- Der Schlüssel columnspan kann Spalten zusammenfassen.

# Schaltfächen (Buttons)

- Starten von Aktionen.
- Der Text auf einer Schaltfläche erläutert die Aktion, die ausgeführt wird.
- Die Aktion kann auch mit einem Icon erläutert werden.

# … erzeugen

```
self.cmdZeichnen = Button(self,text='zeichnen')
```

- Mit Hilfe der Initialisierungsmethode Button() wird eine Schaltfläche erzeugt.
- Die Schaltfläche wird in dem Platzhalter self () positioniert. Das Widget wird in seinem Eltern-Container platziert.
- Der Verweis auf die erzeugte Schaltfläche wird in der Instanzvariablen cmdZeichnen gespeichert.

# Beschriftung einer Schaltfläche

```
self.cmdZeichnen = Button(self,text='zeichnen')
```

- Dem Schlüssel text wird eine Beschriftung für die Schaltfläche zugewiesen.
- Die Aktion wird häufig durch ein Verb beschrieben.

# Weitere Eigenschaften

```
self.cmdZeichnen = Button(self,text='zeichnen')
self.cmdZeichnen['text'] = beschriftung
self.cmdZeichnen['width'] = 30
self.cmdZeichnen['font'] = ('Arial',18, 'bold')
```

- Der Schlüssel wird in eckigen Klammern als String angegeben.
- Mit Hilfe des Gleichheitszeichen wird den Schüsseln ein passender Wert zugewiesen.

# Anbindung an eine Aktion

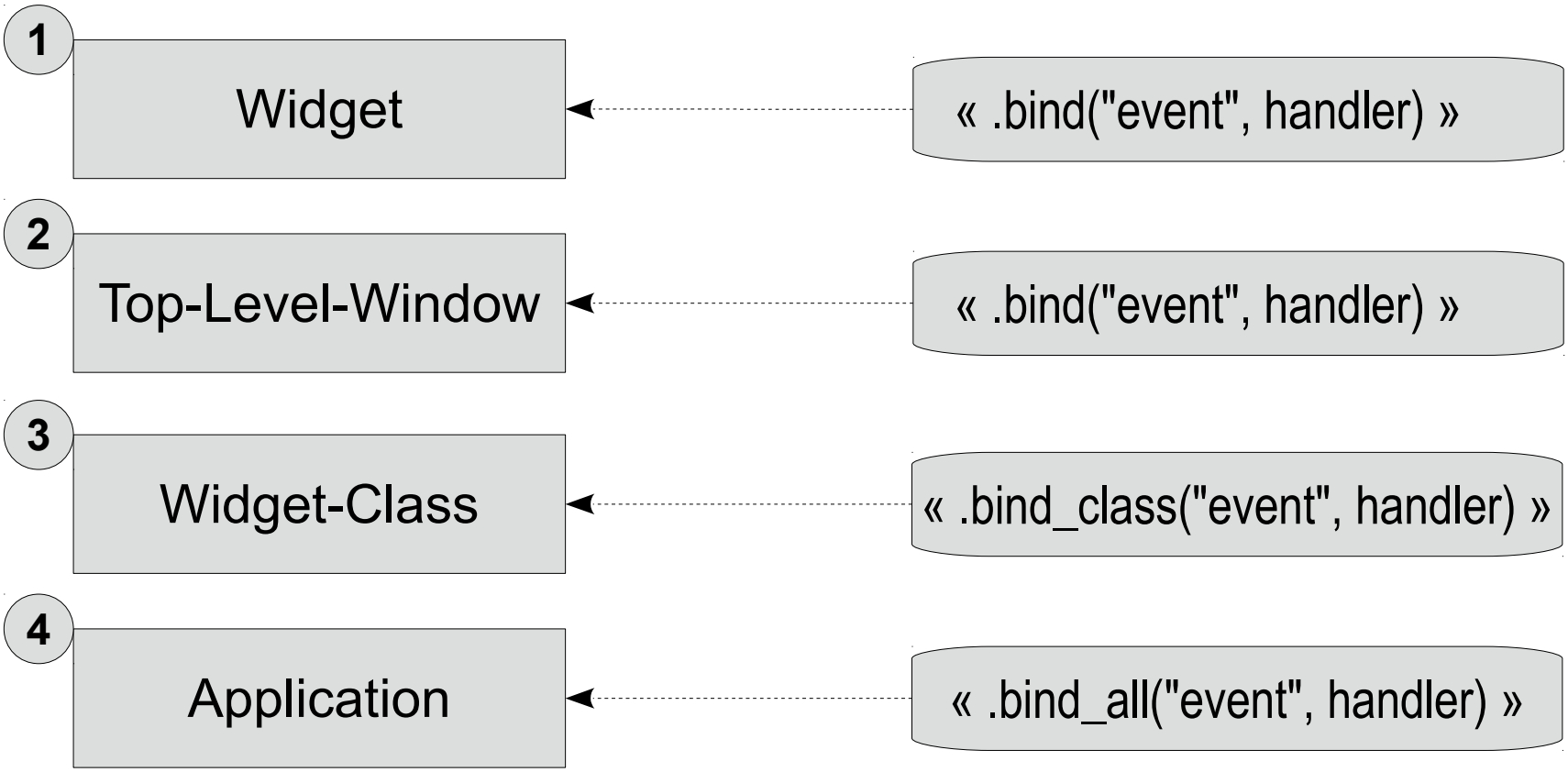
```
self.cmdZeichnen.bind("<Button-1>", self.zeichnen)
```

- Die Methode bind() verbindet eine Schaltfläche mit der auszuführenden Aktion.
- Der erste Parameter beschreibt das Ereignis, welches die Aktion auslöst.
- Die auszuführende Aktion wird in einer Methode beschrieben. Der zweite Parameter ruft die Methode auf. Die Klammern für die Parameterliste werden nicht gesetzt. Dem zweiten Parameter wird ein Event-Handler übergeben.

# Ereignis (Event)

- Auslösung durch ein Fenster, Container oder Widget.
- Benutzeraktionen wie „Drücken einer Maustaste“, „“Drücken einer Taste“ oder „Neukonfiguration des Fensters“ sind Ereignisse
- Die Reaktion auf Ereignisse kann durch eine Methode abgebildet werden, muss aber nicht.

# Abarbeitung von Ereignissen



# Event-Handler

```
def zeichnen(self, event):
    self.zeichenflaeche.delete("all")
    self.zeichneKoordinatenkreuz()

    for zaehler in range(4000):
        x = zaehler / 200.0 - 10
        formel = eval(self.txtFormel.get())
        self.zeichenflaeche.create_line(x * 20 + 200,
                                        200 - formel * 20,
                                        x * 20 + 201,
                                        200 - formel * 20)
```

# Kopf eines Event-Handlers

```
def zeichnen(self, event):
```

- Der Parameter self beschreibt das Objekt, welches das Ereignis ausgelöst hat.
- Der Parameter event beschreibt das Ereignis, welches ausgelöst wurde.

# Regeln für den Namen eines Events

- Der Bezeichner kann sich aus den Buchstaben a...z, A...Z, die Zahlen 0...9 und den Unterstrich zusammensetzen.
- Der Name der Methode beginnt mit einem Kleinbuchstaben.
- Es wird die Groß- und Kleinschreibung beachtet.
- Der Name von Methoden ist in einer Klasse eindeutig.

# Hinweise zur Namensgebung

- Der Name sollte die Aktion beschreiben.
- Der Name kann aus dem Namen des Auslöser sowie der Event-Bezeichnung zusammengesetzt werden.

# Neukonfiguration eines Fensters

```
self.bind("<Configure>", self.zeichnen)
```

- Auslösung bei Neuzeichnung eines Fenster.
- Größenanpassungen des Fensters.

# Maus-Ereignisse

- "<Button-1>". Die linke Maustaste ist gedrückt.
- "<Enter>". Der Mauszeiger wird in ein Widget hinein bewegt.
- "<Leave>". Der Mauszeiger verlässt ein Widget.
- "<FocusIn>". Das Widget bekommt den Fokus.
- "<FocusOut>". Das Widget verliert den Fokus.

# Beispiel

```
self.cmdZeichnen.bind("<Button-1>", self.zeichnen)
```

```
# Informationen zum Event
print("Auslöser: ", event.widget)
print("an der Position: ", event.x, " - ", event.y)
```

# Tastatur-Ereignisse

- "<Key>". Der Benutzer drückt eine Taste.
- "<Return>". Der Benutzer drückt die „Return“-Taste.

# Beispiel „Key“

```
self.txtBenutzername.bind("<Key>", self.pruefeKey)
```

```
# Informationen zum Event
print("Zeichen: ", event.char)
```

# Beispiel „Return“

```
self.txtBenutzername.bind("<Return>", self.sageGutenTag)
```

```
def sageGutenTag(self, event):
        strBenutzername = self.txtBenutzername.get()
        self.lblHallo['text'] = "Guten Tag " + strBenutzername
```

# Informationen aus einem Textfeld

```
strBenutzername = self.txtBenutzername.get()
```

- Die Methode get() gibt den Inhalt eines Textfeldes zurück.
- Der Inhalt wird immer als String zurück gegeben und muss mit den entsprechenden Funktionen wie zum Beispiel float() konvertiert werden.