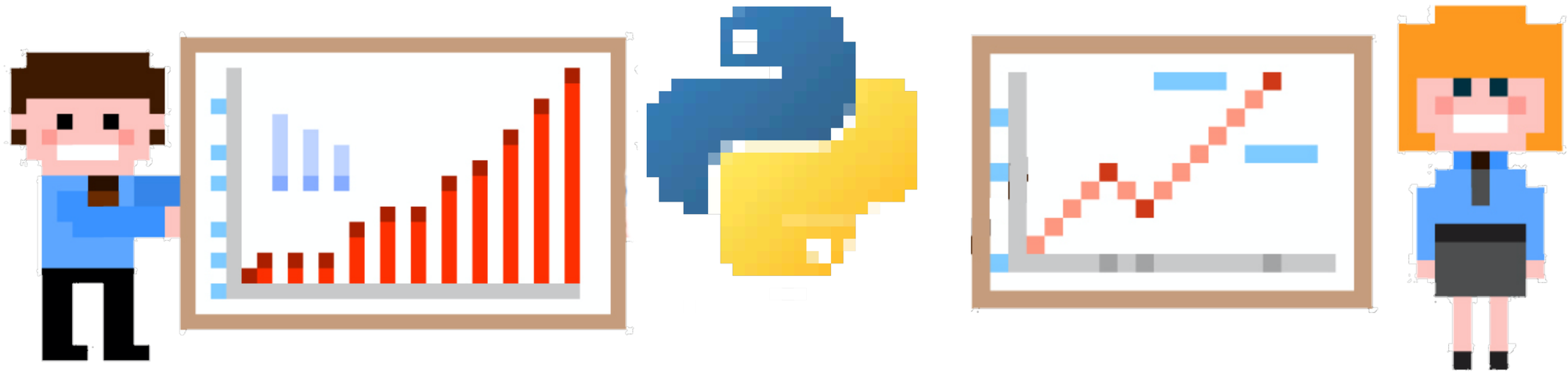
Dashboards und Apps with Dash



Stefan Baerisch, stbaer.com, enterPy 2021

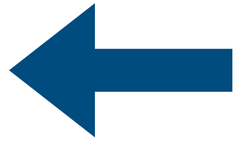
Image: Freepic Professional

Was ist Dash?

- Bibliothek zum Erstellen von analytischen Webapplikationen
- Python Zugang zu HTML, Komponenten für Diagrammen, und Interaktivität
- Open Source mit kommerziellen Addons

Wofür verwenden?

- Dashboards
- Applikationen und Web GUIs



unser Fokus heute

- Teil von bestehenden Applikationen?

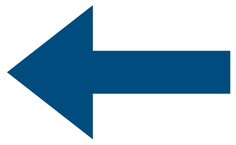
Dash Versionen

Dash Open Source

- Kernbibliothek
- HTML & Bootstrap Komponenten
- Input Komponenten
- Diagramme
- Einbettung, Authentifizierung, etc. in Handarbeit

Dash Enterprise

- Deployment
- Authentification
- Einbettung in Applikationen
- PDF Reports
- Designkit



unser Fokus heute

Dash Beispiele

Architektur

Anwendung

Python

Input Components

Core Components

HTML Components

Implementierung

HTML

React

Flask

Eine Website in Dash

```
from dash import Dash
import dash_html_components as html


app = Dash(__name__)

app.layout = html.Div("Hello World")

if __name__ == '__main__':
    app.run_server(debug=True)
```

Ausgabe: Eine Website in Dash

Hello World

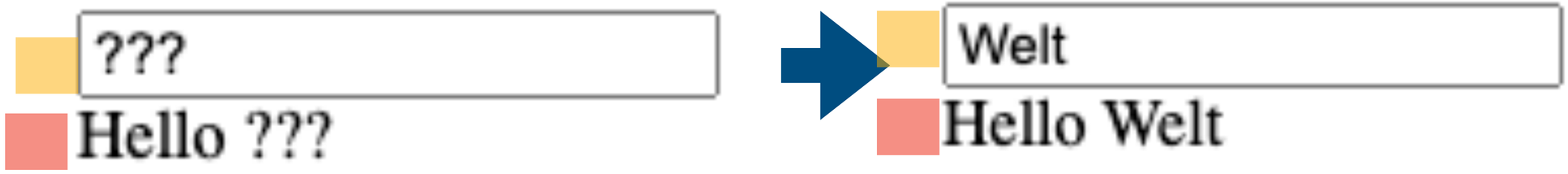
```
<!DOCTYPE html>
<html>
▶ <head>…</head>
▼ <body>
  ▼ <div id="react-entry-point">
    ▼ <div id="_dash-global-error-container">
      ▼ <div>
        ▶ <div class="dash-debug-menu__outer dash-debug-menu__outer--closed">…
          </div> flex
        ▶ <div class="dash-debug-menu dash-debug-menu--closed">…</div> flex
      ⋯ ▼ <div> == $0
          ▼ <div>
            ▼ <div id="_dash-app-content">
                <div>Hello World</div>
              </div>
            </div>
          ▶ <div class="dash-error-menu">…</div>
          </div>
        </div>
      </div>
    </div>
  ▶ <footer>…</footer>
  </body>
</html>
```

Callbacks - Auf Eingaben reagieren

```
import dash_core_components as dcc
import dash_html_components as html
from dash import Dash
from dash.dependencies import Input, Output

app = Dash(__name__)

app.layout = html.Div([
    dcc.Input(id='in1', value='???', type='text'),
    html.Div(id='out1')
])

@app.callback(
    Output(component_id='out1', component_property='children'),
    Input(component_id='in1', component_property='value')
)
def update_name(input_value):
    return f"Hello {input_value}"

if __name__ == '__main__':
    app.run_server(debug=True)
```

Ausgabe: Callbacks - Auf Eingaben reagieren

```
@app.callback(
    Output('out1', 'children'),
    Input('in1', 'value')
)
def update_name(input_value):
```

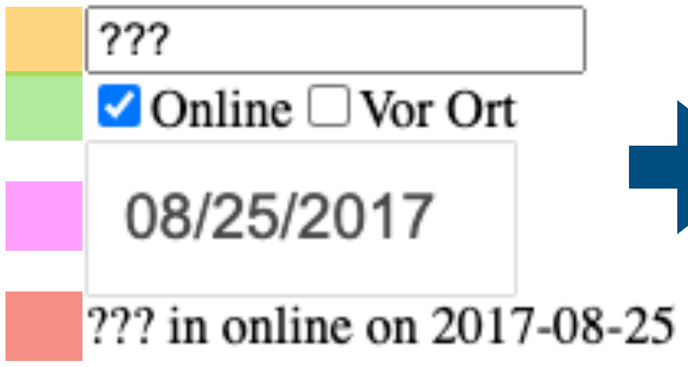


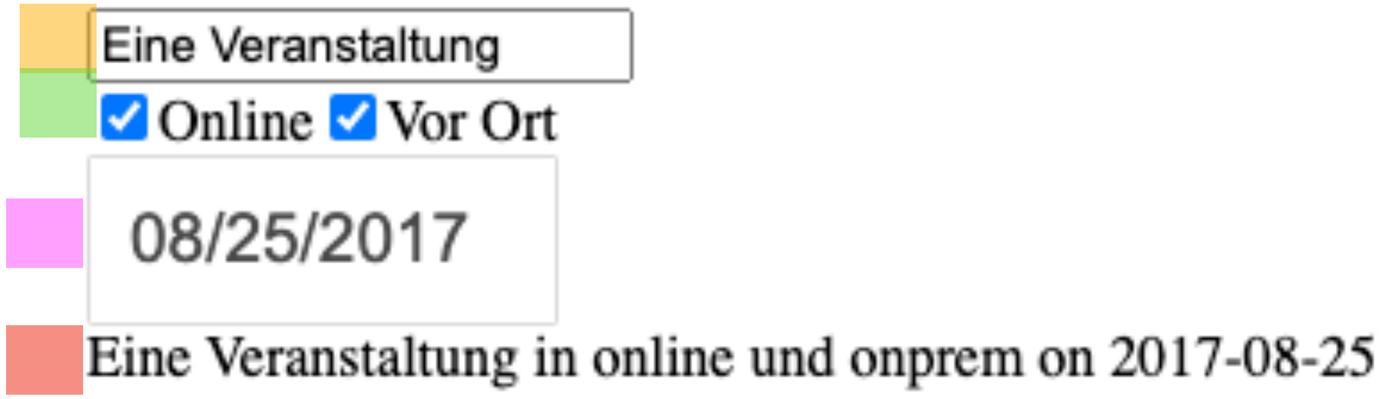
Callback mit mehreren Inputs

```
checklist = dcc.Checklist(
    id='checklist',
    options=[
        {'label': 'Online', 'value': 'online'},
        {'label': 'Vor Ort', 'value': 'onprem'},
    ],
    value=['online'],
)
datepick = dcc.DatePickerSingle(
    id='datepick',
    min_date_allowed=date(1995, 8, 5),
    max_date_allowed=date(2017, 9, 19),
    initial_visible_month=date(2017, 8, 5),
    date=date(2017, 8, 25)
)
app.layout = html.Div([
    dcc.Input(id='in1', value='???', type='text'),
    checklist,
    datepick,
    html.Div(id='out1')
])

@app.callback(
    Output(component_id='out1', component_property='children'),
    Input(component_id='in1', component_property='value'),
    Input(component_id='checklist', component_property='value'),
    Input(component_id='datepick', component_property='date')
)
def update_name(text, check, date):
    check_text = " und " .join(check)
    return f"{text} in {check_text} on {date}"
```

Ausgabe: Callback mit mehreren Inputs

```
@app.callback(
    Output('out1', 'children'),
    Input('in1', 'value'),
    Input('checklist', 'value'),
    Input('datepick', 'date')
)
def update_name(text, check, date):
```



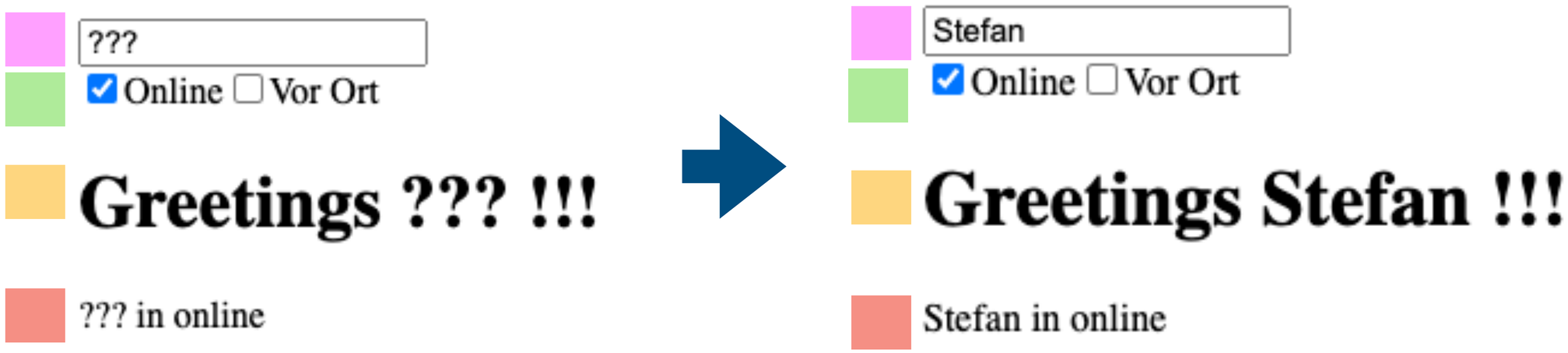


Callback mit mehreren Outputs

```
checklist = dcc.Checklist(
    id='checklist',
    options=[
        {'label': 'Online', 'value': 'online'},
        {'label': 'Vor Ort', 'value': 'onprem'},

    ],
    value=['online'],
)
app.layout = html.Div([
    dcc.Input(id='in1', value='???', type='text'),
    checklist,
    html.H1(id='header_1'),
    html.Div(id='div_1')
])
@app.callback(
    Output(component_id='div_1', component_property='children'),
    Output(component_id='header_1', component_property='children'),
    Input(component_id='in1', component_property='value'),
    Input(component_id='checklist', component_property='value'),
)
def update_name(text, check):
    check_text = " und ".join(check)
    div_text = f"{text} in {check_text}"
    h1_text = f"Greetings {text} !!!"
    return div_text, h1_text
```

Callback mit mehreren Outputs

```
@app.callback(
    Output('div_1', 'children'),
    Output('header_1', 'children'),
    Input('in1', 'value'),
    Input('checklist', 'value'),
)
def update_name(text, check):
```



Callbacks mit abhängigen Inputs

```
df = pd.read_excel('./staedte.xlsx')


def list_to_options(options):
    return [{'label': o, 'value': o} for o in options]


app.layout = html.Div([
    dcc.Checklist(
        id='cont',
        options=list_to_options(df['Kontinent'].unique())),
    dcc.Checklist(id='nation', options=[]),
    dcc.Checklist(id='city', options=[]),
    html.Div(id='div_1')
])

@app.callback(
    Output('nation', 'options'),
    Output('nation', 'value'),
    Input('cont', 'value')
)
def continent_callback(vals):
    df2 = df[df['Kontinent'].isin(vals or [])]
    v  =df2['Staat'].unique()
    return list_to_options(v),v
```

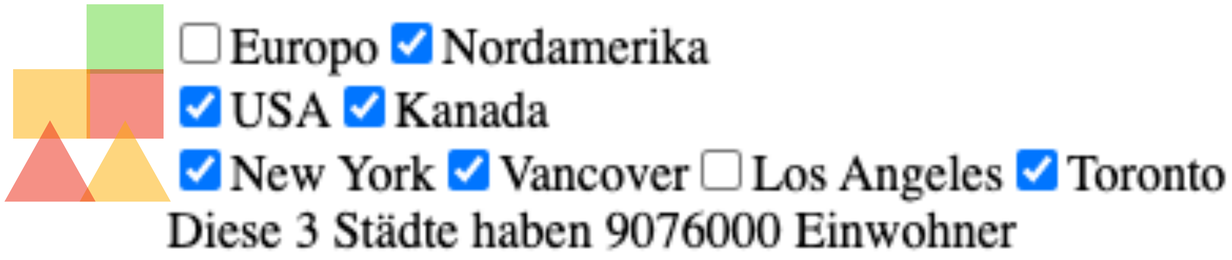
```
@app.callback(
    Output('city', 'options'),
    Output('city', 'value'),
    Input('nation', 'value')
)
def city_callback(vals):
    df2 = df[df['Staat'].isin(vals or [])]
    v = df2['Stadt'].unique()
    return list_to_options(v),v

@app.callback(
    Output('div_1', 'children'),
    Input('city', 'value'),
)
def city_callback(vals):
    df2 = df[df['Stadt'].isin(vals or [])]
    s = df2['Einwohner'].sum()
    return f"Diese {len(df2)} Städte haben {s} Einwohner"
```

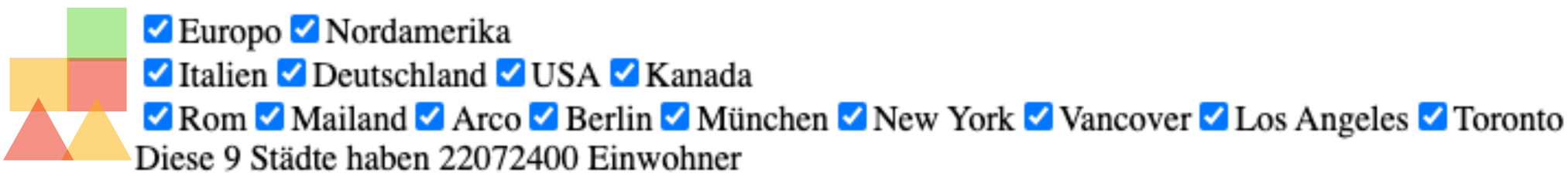
Ausgabe: Callbacks mit abhängigen Inputs

```
@app.callback(
    Output('nation', 'options'),
    Output('nation', 'value'),
    Input('cont', 'value')
)
continent_callback(vals):
```

```
@app.callback(
    Output('city', 'options'),
    Output('city', 'value'),
    Input('nation', 'value')
)
def city_callback(vals):
```

```
@app.callback(
    Output('div_1', 'children'),
    Input('city', 'value'),
)
def city_callback(vals):
```



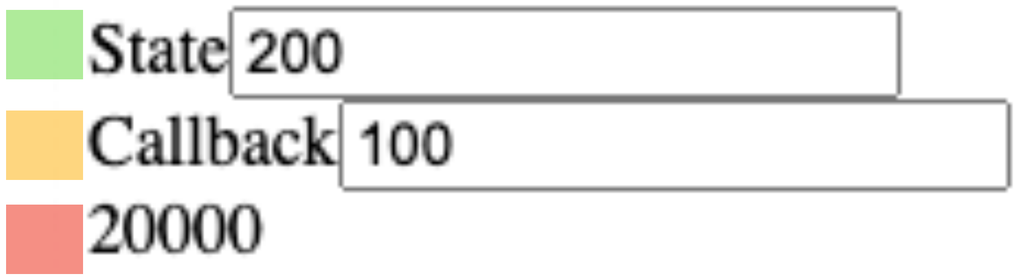


Callbacks und Status

```
app.layout = html.Div([
    html.Div([
        html.Span(children="State"),
        dcc.Input(id='state', type='text', value='10'),
    ]),
    html.Div([
        html.Span(children="Callback"),
        dcc.Input(id='call', type='text', value='10'),
    ]),

    html.Div(id='out')
])

@app.callback(Output('out', 'children'),
              Input('call', 'value'),
              State('state', 'value'))
def update_output(call, state):
    try:
        return int(call) * int(state)
    except:
        "Error"
```

Ausgabe: Callbacks und Status

```
@app.callback(
    Output('out', 'children'),
    Input('call', 'value'),
    State('state', 'value'))
def update_output(call, state):
```

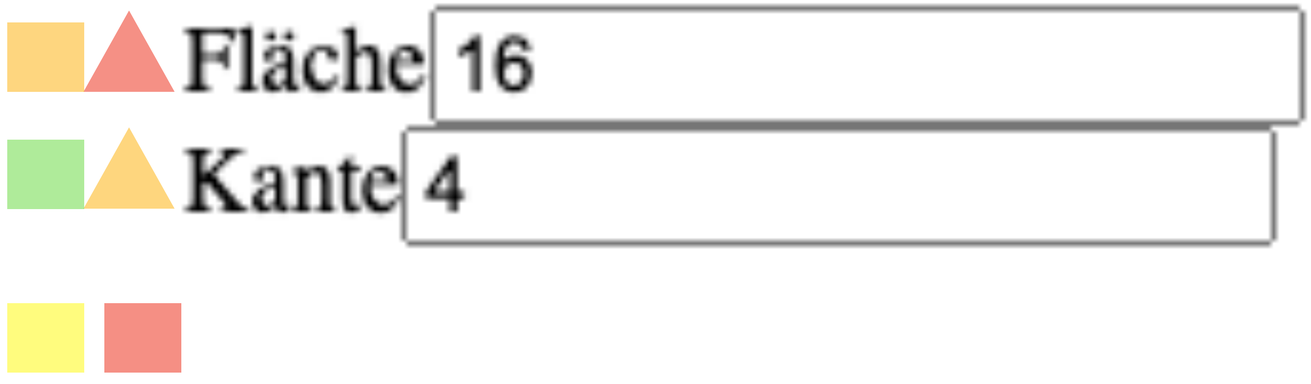


Callbacks mit Input = Output

```
app.layout = html.Div([
    html.Div([
        html.Span(children="Fläche"),
        dcc.Input(id='area', type='text', value='100'),
    ]),
    html.Div([
        html.Span(children="Kante"),
        dcc.Input(id='length', type='text', value='10'),
    ]),
    html.Div(id="state",style={"display":"none"},children="100")
])

@app.callback(Output("area", 'value'),
              Output("length", 'value'),
              Output("state", "children"),
              Input("area", 'value'),
              Input("length", 'value'),
              State("state", "children")
              )
def update_output(area,length,state):
    area = float(area)
    length = float(length)
    area_old = float(state)
    if area != area_old:
        length = math.sqrt(area)
    else:
        area = length * length
    return area,length,area
```

Ausgabe: Callbacks mit Input = Output

```
@app.callback(
    Output("area", 'value'),
    Output("length", 'value'),
    Output("state", "children"),
    Input("area", 'value'),
    Input("length", 'value'),
    State("state", "children")
              )
def update_output(area,length,state):
```

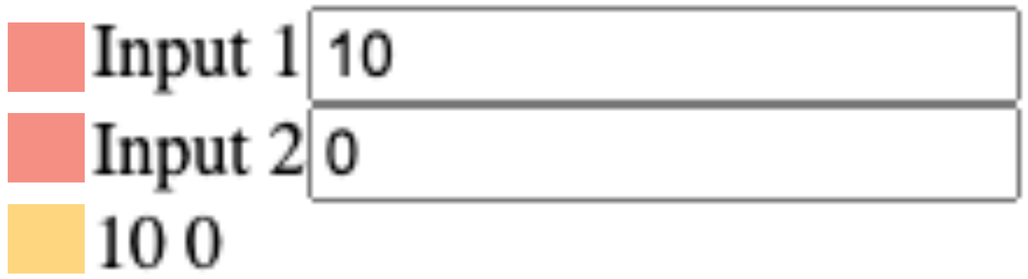


Callbacks mit Pattern Matching

```
app.layout = html.Div([
    html.Div([
        html.Span(children="Input 1"),
        dcc.Input(id={'type': 'text-input','index': 1}, type='text', value='0'),
    ]),
    html.Div([
        html.Span(children="Input 2"),
        dcc.Input(id={'type': 'text-input','index': 2}, type='text', value='0'),
    ]),
    html.Div(id="text-output")
])


@app.callback(
    Output("text-output", 'children'),
    Input({'type': 'text-input', 'index': ALL}, 'value')
)
def update(values):
    return " ".join(values)
```

Ausgabe: Callbacks mit Pattern Matching

```
@app.callback(
    Output("text-output", 'children'),
    Input({'type': 'text-input', 'index': ALL}, 'value')
)
def update(values):
```



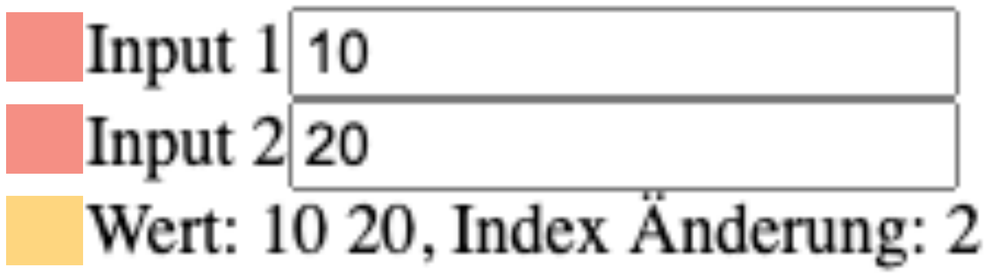
Quellelement für Callbacks identifizieren

```
app.layout = html.Div([
    html.Div([
        html.Span(children="Input 1"),
        dcc.Input(id={'type': 'text-input', 'index': 1}, type='text', value='0'),
    ]),
    html.Div([
        html.Span(children="Input 2"),
        dcc.Input(id={'type': 'text-input', 'index': 2}, type='text', value='0'),
    ]),
    html.Div(id="text-output"),
])
```

```
@app.callback(
    Output("text-output", 'children'),
    Input({'type': 'text-input', 'index': ALL}, 'value')
)
def update(values):
    ctx = dash.callback_context
    change = ""
    if ctx:
        change_json = ctx.triggered[0]['prop_id'].split('.')[0]
        if change_json:
            change = json.loads(change_json)['index']
    values = " ".join(values)
    return f"Wert: {values}, Index Änderung: {change}"
```

Ausgabe: Quellelement für Callbacks identifizieren

```
@app.callback(
    Output("text-output", 'children'),
    Input({'type': 'text-input', 'index': ALL}, 'value')
)
def update(values):
```



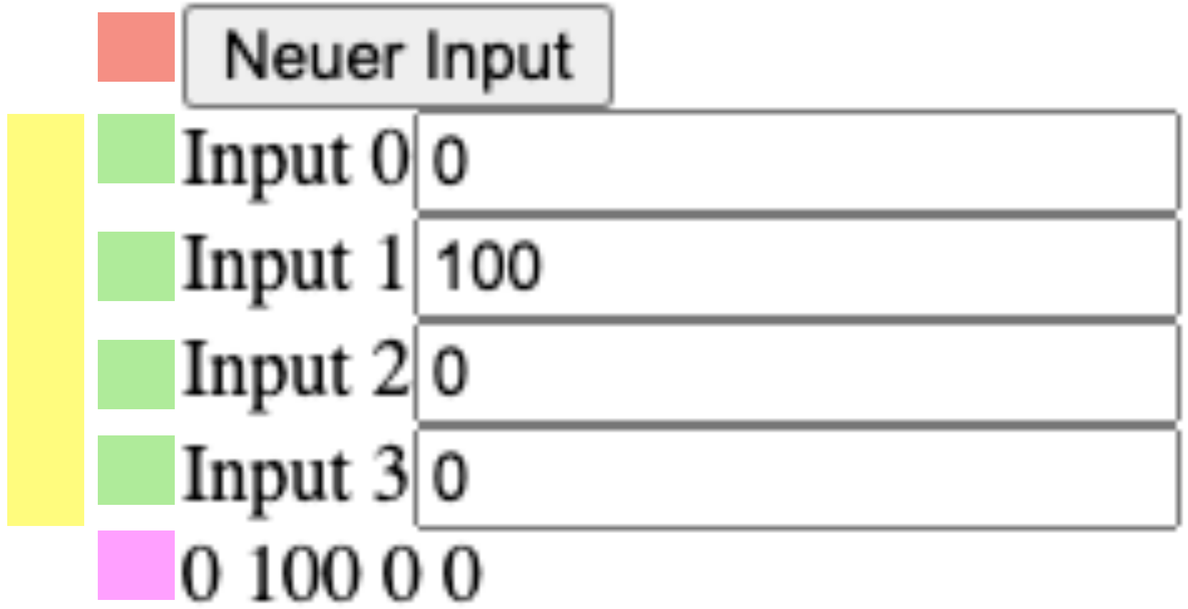
Dynamische Elemente

```
app.layout = html.Div([
    html.Button("Neuer Input", id="neu", n_clicks=0),
    html.Div(id='inputs', children=[]),
    html.Div(id='result')
])

@app.callback(
    Output('inputs', 'children'),
    Input('neu', 'n_clicks'),
    State('inputs', 'children'))
def add_input(n_clicks, children):
    entry = html.Div([
        html.Span(children=f"Input {n_clicks}"),
        dcc.Input(id={'type': 'text-input', 'index': n_clicks}, type='text', value='0'),
    ])
    children.append(entry)
    return children

@app.callback(
    Output("result", 'children'),
    Input({'type': 'text-input', 'index': ALL}, 'value'))
def update(values):
    return " ".join(values)
```

Ausgabe: Dynamische Elemente

```
@app.callback(
    Output('inputs', 'children'),
    Input('neu', 'n_clicks'),
    State('inputs', 'children'))
def add_input(n_clicks, children):
```

```
@app.callback(
    Output("result", 'children'),
    Input({'type': 'text-input', 'index': ALL}, 'value'))
def update(values):
```



Dynamische Elemente 2

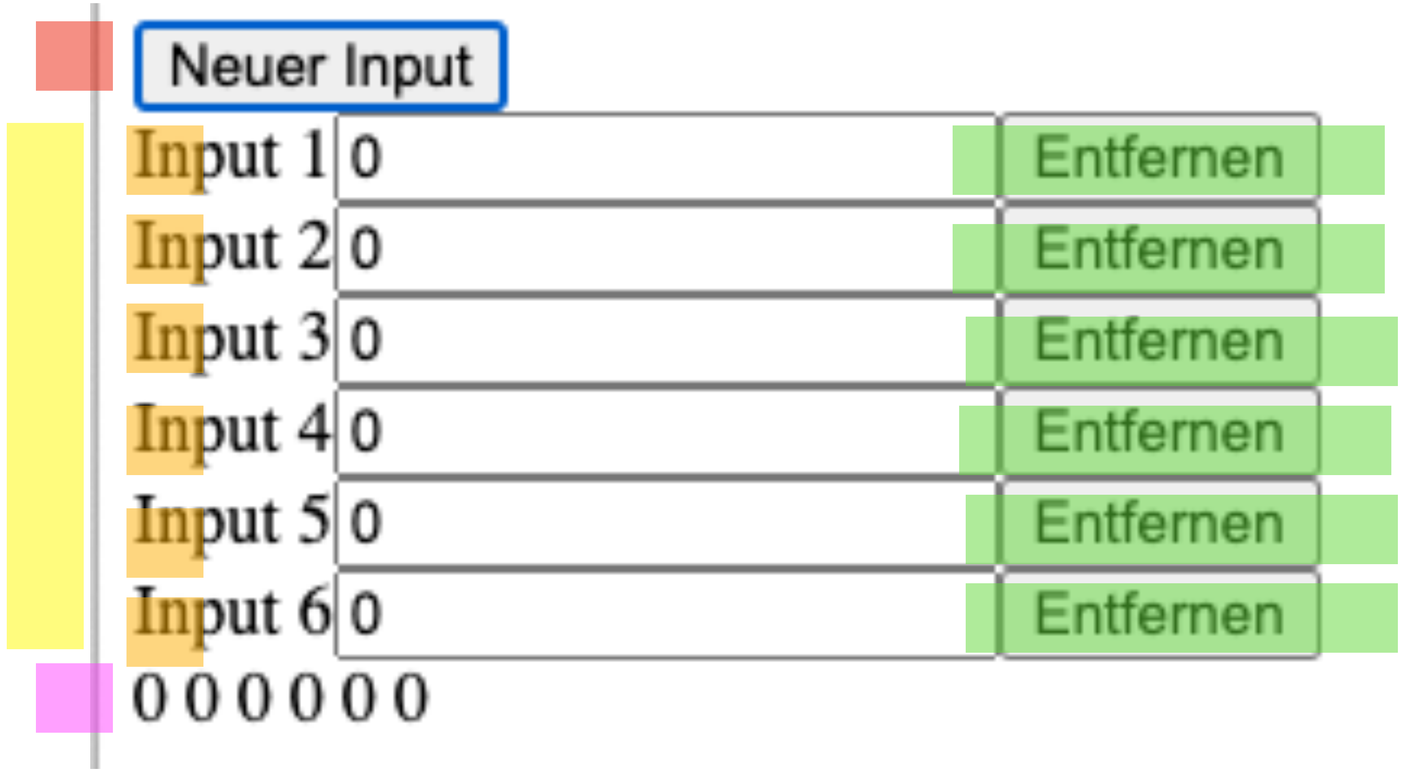
```
app.layout = html.Div([
    html.Button("Neuer Input", id="neu-btn", n_clicks=0),
    html.Div(id='inputs', children=[]),
    html.Div(id='result')
])
```

```
@app.callback(
    Output('inputs', 'children'),
    Input("neu-btn", 'n_clicks'),
    Input({'type': 'btn', 'index': ALL}, 'n_clicks'),
    State('inputs', 'children'), prevent_initial_call=True)
def add_input(n_clicks, _, children):
    triggered = [t["prop_id"] for t in dash.callback_context.triggered][0]
    if triggered == 'neu-btn.n_clicks':
        entry = html.Div(id=f"div-{n_clicks}", children=[
            html.Span(children=f"Input {n_clicks}"),
            dcc.Input(id={'type': 'text-input', 'index': n_clicks},
                      type='text', value='0'),
            html.Button("Entfernen",
                        id={'type': 'btn', 'index': n_clicks}, n_clicks=0),
        ])
        children.append(entry)
        print(_)
    else:
        rx = r'\{"index":(\d+),"type":"btn"\}\.n_clicks'
        want_idx = "div-" + re.match(rx, triggered).group(1)
        children_idxs = [c['props']['id'] for c in children]
        idx = children_idxs.index(want_idx)
        del children[idx]
        print(idx, _)

    return children

@app.callback(
    Output("result", 'children'),
    Input({'type': 'text-input', 'index': ALL}, 'value'))
def update(values):
    return " ".join(values)
```

Ausgabe: Dynamische Elemente 2

```
@app.callback(
    Output('inputs', 'children'),
    Input("neu-btn", 'n_clicks'),
    Input({'type': 'btn', 'index': ALL}, 'n_clicks'),
    State('inputs', 'children'), prevent_initial_call=True)
def add_input(n_clicks, _, children):
```

```
@app.callback(
    Output("result", 'children'),
    Input({'type': 'text-input',
           'index': ALL}, 'value'))
def update(values):
```



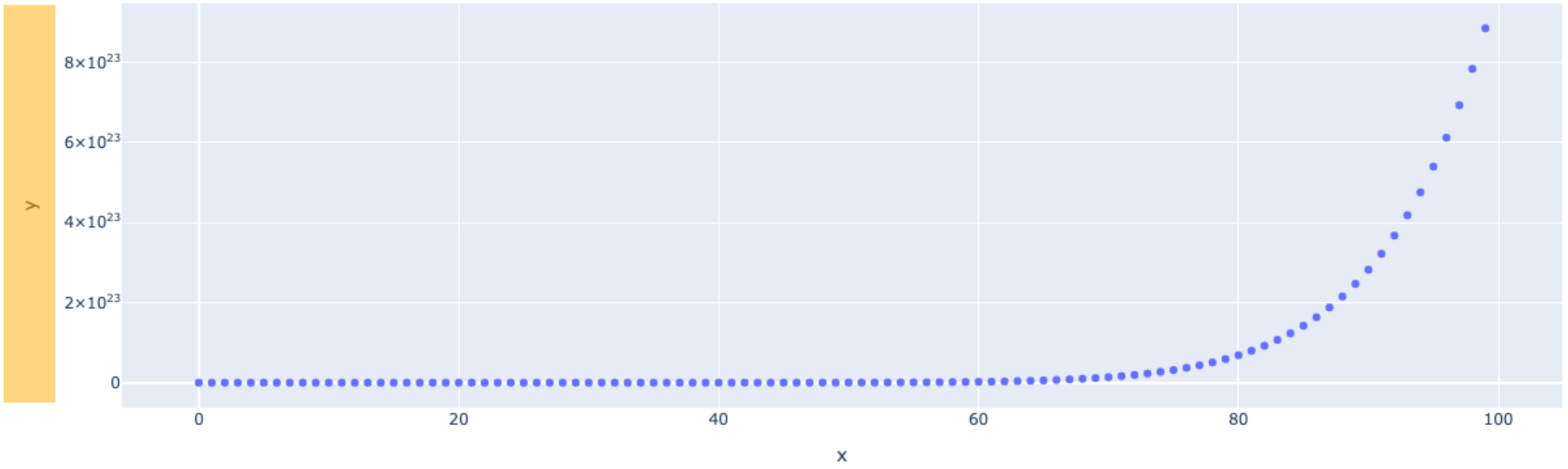
Graphen verwenden

```
import plotly.express as px


 app.layout = html.Div([
     dcc.Input(id='exp', value='1', type='number'),
     dcc.Graph(id='out'),
 ])

 @app.callback(
     Output('out', 'figure'),
     Input('exp', 'value')
 )
 def update_name(value):
     if value is None:
         value = 1
     fig = px.scatter(x=range(100), y=[i ** int(value) for i in range(100)])
     return fig
```

Ausgabe: Graphen verwenden

```
@app.callback(
    Output('out', 'figure'),
    Input('exp', 'value')
)
def update_name(value):
```

12

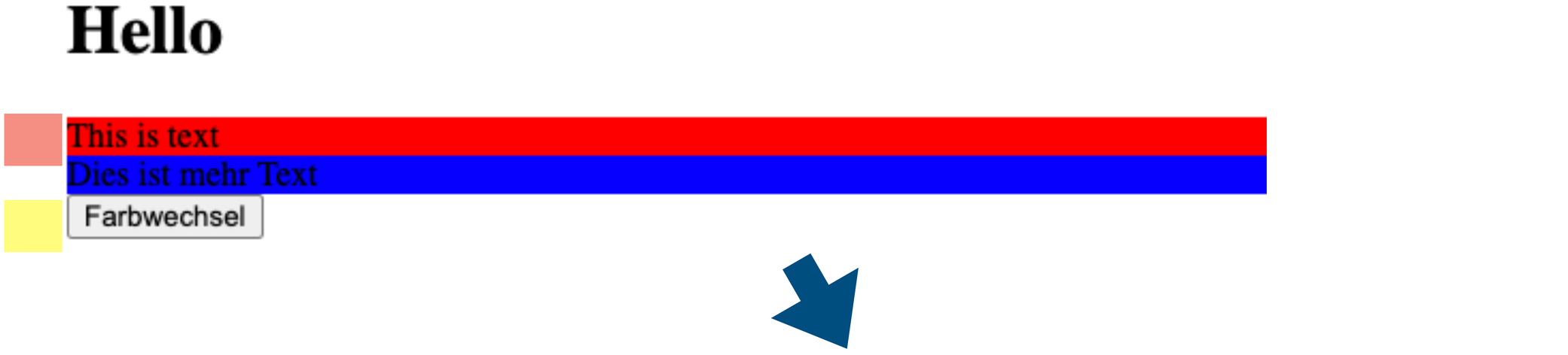


Styling und CSS in Python

```
app.layout = html.Div([
    html.Div([
        html.H1(children="Hello"),
        html.Div(children="This is text", className='div-1'),
        html.Div(id='sdiv', children="Dies ist mehr Text",
         style={
            'background-color': 'green'
             }),
        html.Button("Farbwechsel", id="btn", n_clicks=0),
    ])
])

@app.callback(
    Output('sdiv','style'),
    Input('btn','n_clicks'),
    State('sdiv','style')
)
def update_style(btn, style):
    color = 'blue' if style['background-color'] == 'green' else 'green'
    return {'background-color': color}
```

Ausgabe: Styling und CSS in Python

```
@app.callback(
    Output('sdiv','style'),
    Input('btn','n_clicks'),
    State('sdiv','style')
)
def update_style(btn, style):
```

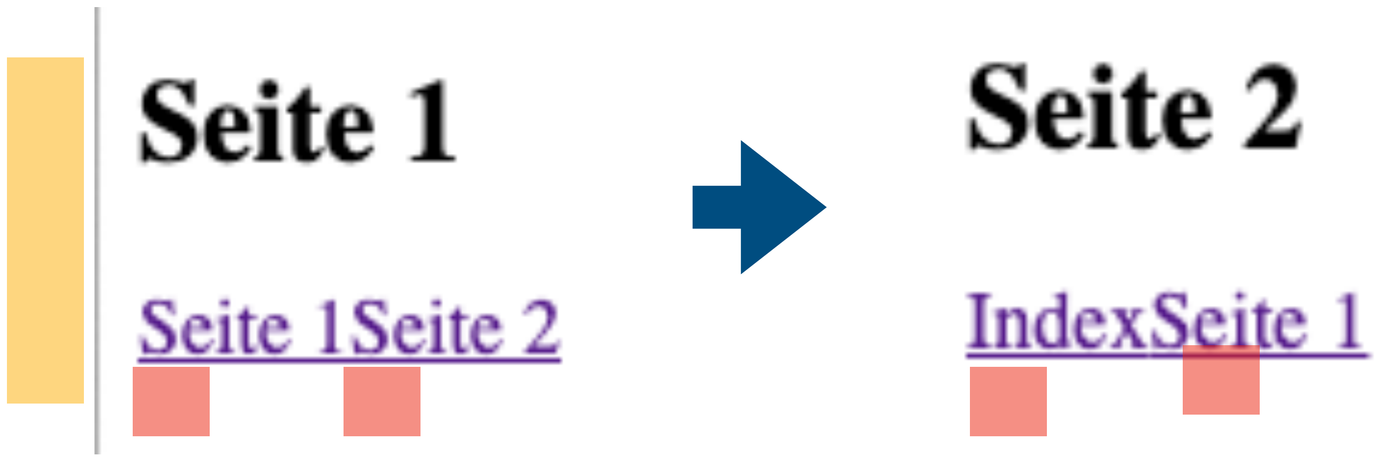


Dash Applikationen mit mehreren Seiten

```
main = html.Div([
    dcc.Location(id='url', refresh=False),
    html.Div(id='main')
])

index = html.Div([
    html.H2("Index"),
    dcc.Link('Seite 1', href='/s1'),
    dcc.Link('Seite 2', href='/s2'),
])

page_1 = html.Div([
    html.H2("Seite 1"),
    dcc.Link('Index', href='/'),
    dcc.Link('Seite 2', href='/s2'),
])

page_2 = html.Div([
    html.H2("Seite 2"),
    dcc.Link('Index', href='/'),
    dcc.Link('Seite 1', href='/s1'),
])

app.layout = main
app.validation_layout = html.Div([main, index, page_1, page_2])

@app.callback(Output('main', 'children'), Input('url', 'pathname'))
def nav(pathname):
    return {'/': index, '/s1': page_1, '/s2': page_2}[pathname]
```

Ausgabe: Dash Applikationen mit mehreren Seiten

```
@app.callback(
    Output('main', 'children'),
    Input('url', 'pathname'))
```

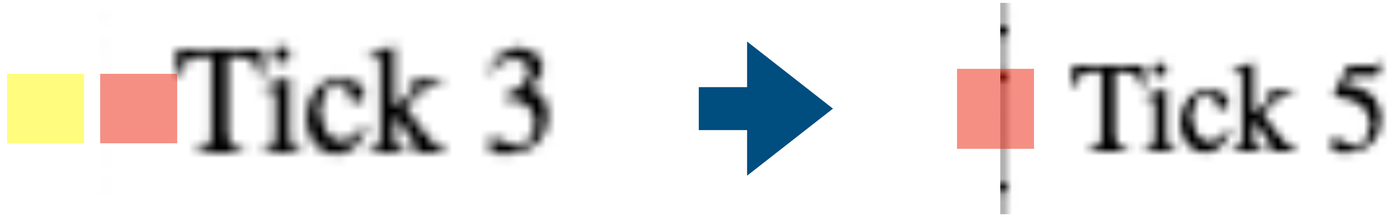


Applikationen automatischen Updates

```
app = dash.Dash(__name__)
app.layout = html.Div(
    html.Div([
        html.Div(id='ticker'),
        dcc.Interval(id='interval', interval=1000, n_intervals=0)
    ])
)

@app.callback(Output('ticker', 'children'),Input('interval', 'n_intervals'))
def do_tick(n):
    return html.Span(f"Tick {n}")
```

Ausgabe: Applikationen automatischen Updates

```
@app.callback(
    Output('ticker', 'children'),
    Input('interval', 'n_intervals')
)
```

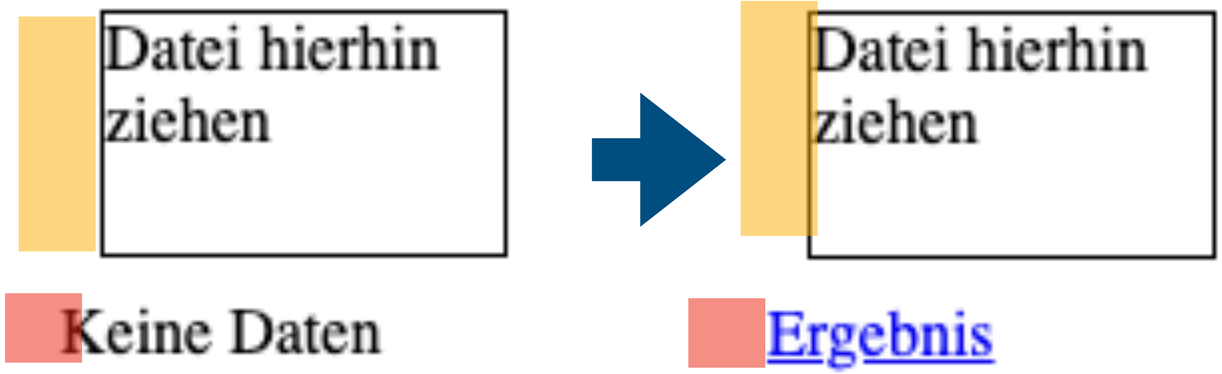


Uploads und Downloads

```
server = Flask(__name__)
app = dash.Dash(server=server)

@server.route("/result")
def download():
    return send_from_directory('uploads', 'result.xlsx', as_attachment=True)

app.layout = html.Div([
    dcc.Upload(id="upload", children=html.Div("Datei hierhin ziehen"),
               style=MY_STYLE),
    html.Div(id="download")])

@app.callback(Output("download", "children"), Input("upload", "contents"))
def update_output(data):
    if data is None:
        return "Keine Daten"
    else:
        header, data = data.encode("utf8").split(b";base64,")
        if not b"openxmlformats-officedocument.spreadsheetml.sheet" in header:
            return "Nur xlsx Datei werden unterstützt"
        df = pd.read_excel(base64.decodebytes(data))
        df['c'] = df['a'] * df['b']
        df.to_excel('uploads/result.xlsx')
        return html.A("Ergebnis", href="/result")

if __name__ == "__main__":
    app.run_server(debug=True, port=9012)
```

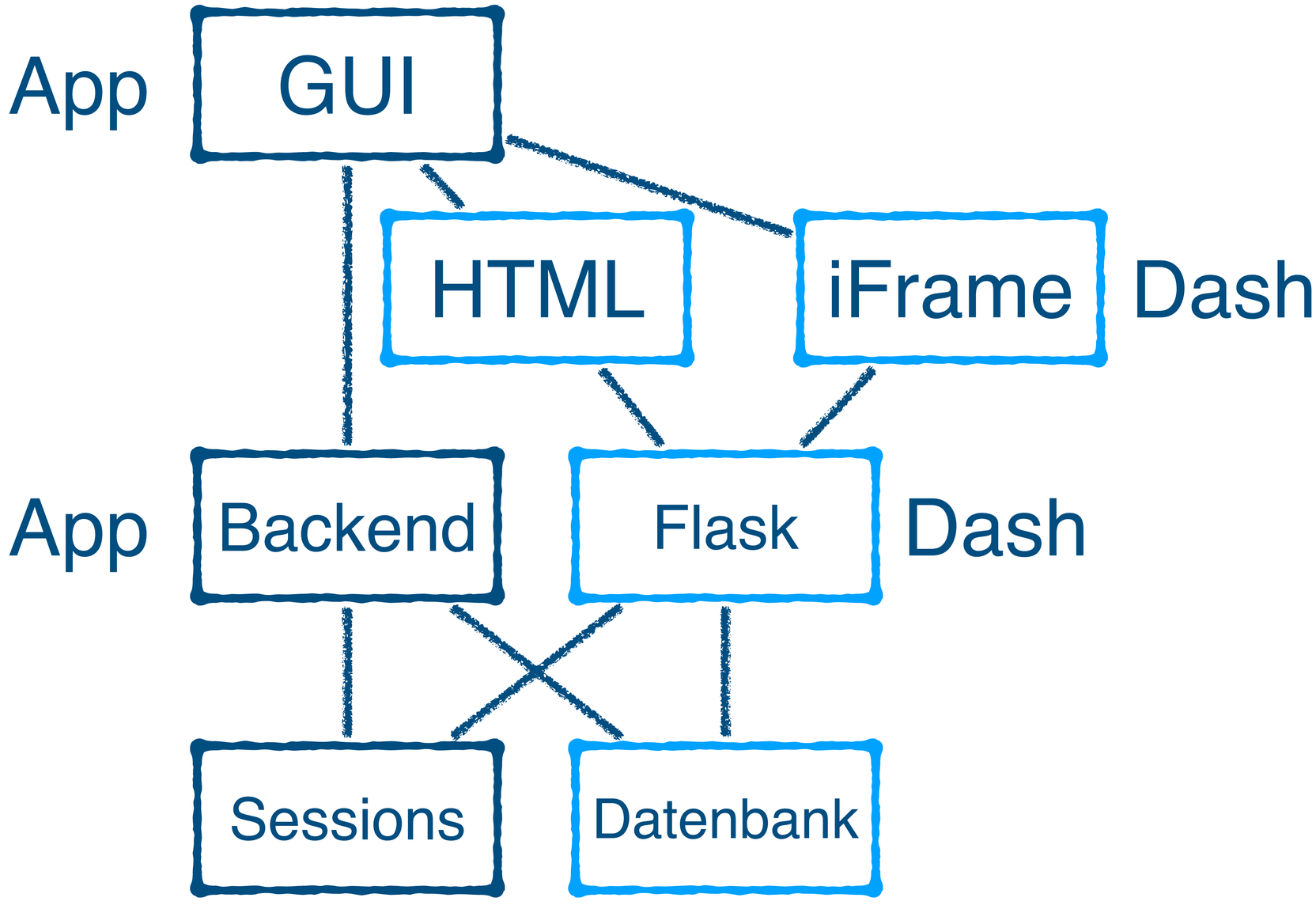
Ausgabe: Uploads und Downloads

```
@app.callback(
    Output("download", "children"),
    Input("upload", "contents")
)
def update_output(data):
```

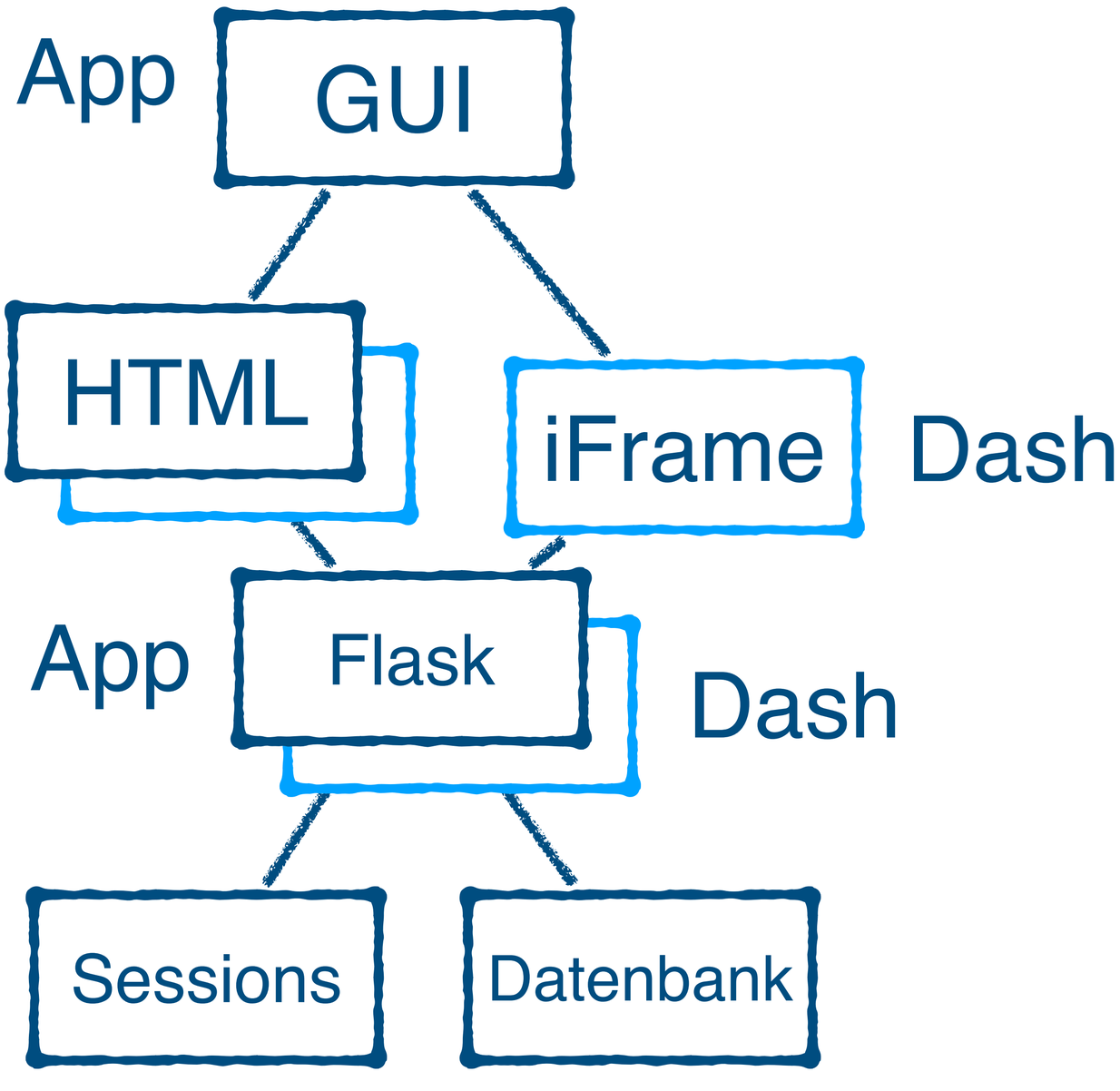


Dash in Applikationen einbinden

Integrationsszenarien



Integrationsszenarien (2)



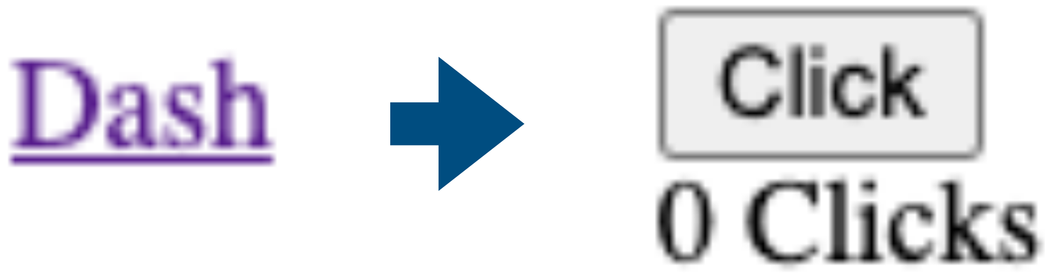
Kombination mit Flask

```
class DashApp:
    def __init__(self, flask_server):
        self.app = dash.Dash(name=self.__class__.__name__,
                             routes_pathname_prefix='/dash/',
                             server=flask_server)

    def setup(self):
        self.setup_layout()
        self.setup_callbacks()

    def setup_layout(self):
        self.app.layout = html.Div([
            html.Button("Click", id='btn'),
            html.Div(id='out')
        ])

    def setup_callbacks(self):
        @self.app.callback(Output('out', 'children'), Input('btn', 'n_clicks'))
        def update(n):
            n = 0 if n is None else n
            return f"{n} Clicks"

app = Flask(__name__)
dash_app = DashApp(app)
dash_app.setup()

@app.route('/')
def index():
    return "<a href='/dash/'>Dash</a>"
```

Ausgabe: Kombination mit Flask

```
@app.route('/')
def index():
    return "<a href='/dash/'>Dash</a>"
```

```
def setup_callbacks(self):
    @self.app.callback(
      Output('out', 'children'),
      Input('btn', 'n_clicks'))
    def update(n):
```

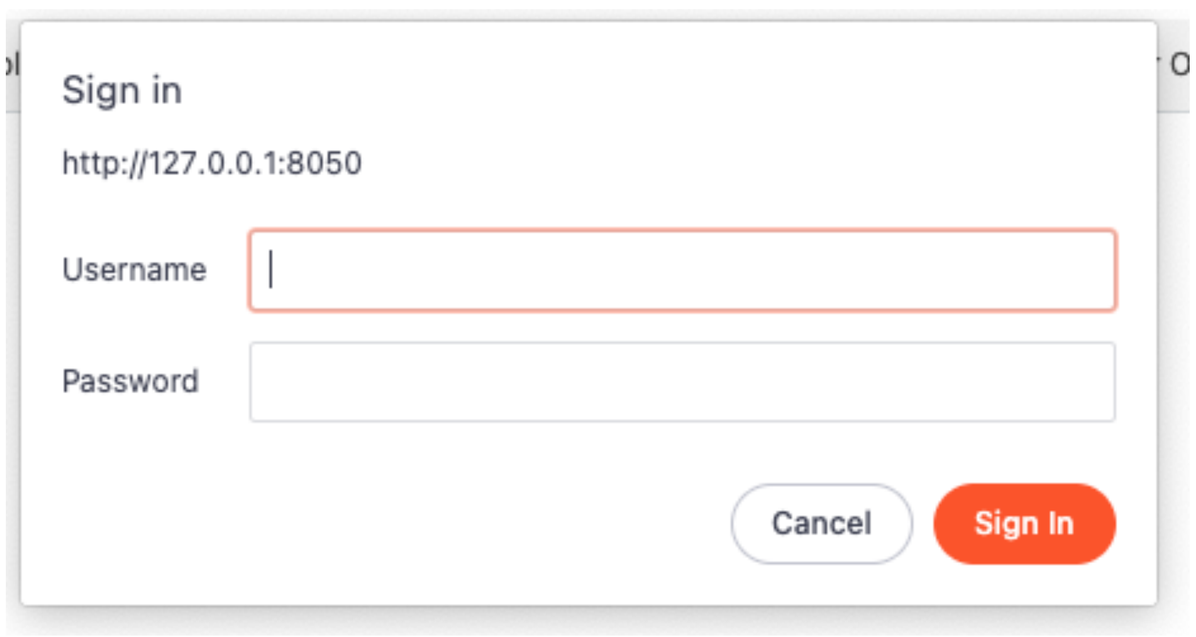


Flask, Dash und Basic-Auth

```
USER = {'user': 'pass'}

app = dash.Dash(__name__)
auth = dash_auth.BasicAuth(app, USER)

app.layout = html.Div([
    dcc.Input(id='in'),
    html.Div(id='out')
])

@app.callback(
    dash.dependencies.Output('out', 'children'),
    dash.dependencies.Input('in', 'value'))
def update_graph(val):
    if val:
        return "".join(reversed(val))
```

Ausgabe: Flask, Dash und Basic-Auth

```
USER = {'user': 'pass'}

app = dash.Dash(__name__)
auth = dash_auth.BasicAuth(app, USER)
```



Einbindung in Django

- Eine Integration mit Django ist über iFrame und gemeinsames Session-Backend möglich
- Mit django-plotly-dash ist der Prozess automatisiert.
- Dash Applikationen (iFrames) werden also Django Template Tags angeboten

Fazit

Fazit

- Die bekannteste Funktion von Dash ist die Erstellung von interaktiven Dashboards
- Das Framework ist jedoch auch dafür geeignet, Oberflächen für Skripte zu erstellen.
- Erweiterung zum Backend lassen sich über Flask Programmierung erzielen
- Durch die Abgrenzung von kommerziellen Komponente gibt es einige Lücken.

Nächste Schritte und Alternativen

- Die Dash Dokumentation bietet einen guten Überblick der verfügbaren Komponenten, wie z.B interaktiven Tabellen.
- Die plotly dash-sample-apps bieten Duzende Beispiele für Applikationen, speziell zur interaktiven Visualisierung
- Bei Interesse an der Entwicklung von Webapplikationen in reinem Python mag auch Anvil interessant sein.

Vielen Dank