

Interaktive Web-Applikationen ohne Javascript mit Dash

Daniel Hepper
pythoncamp 2020

@danielhepper

#Django
#Fullstack
#Freelancer

consideratecode.com



Das Problem

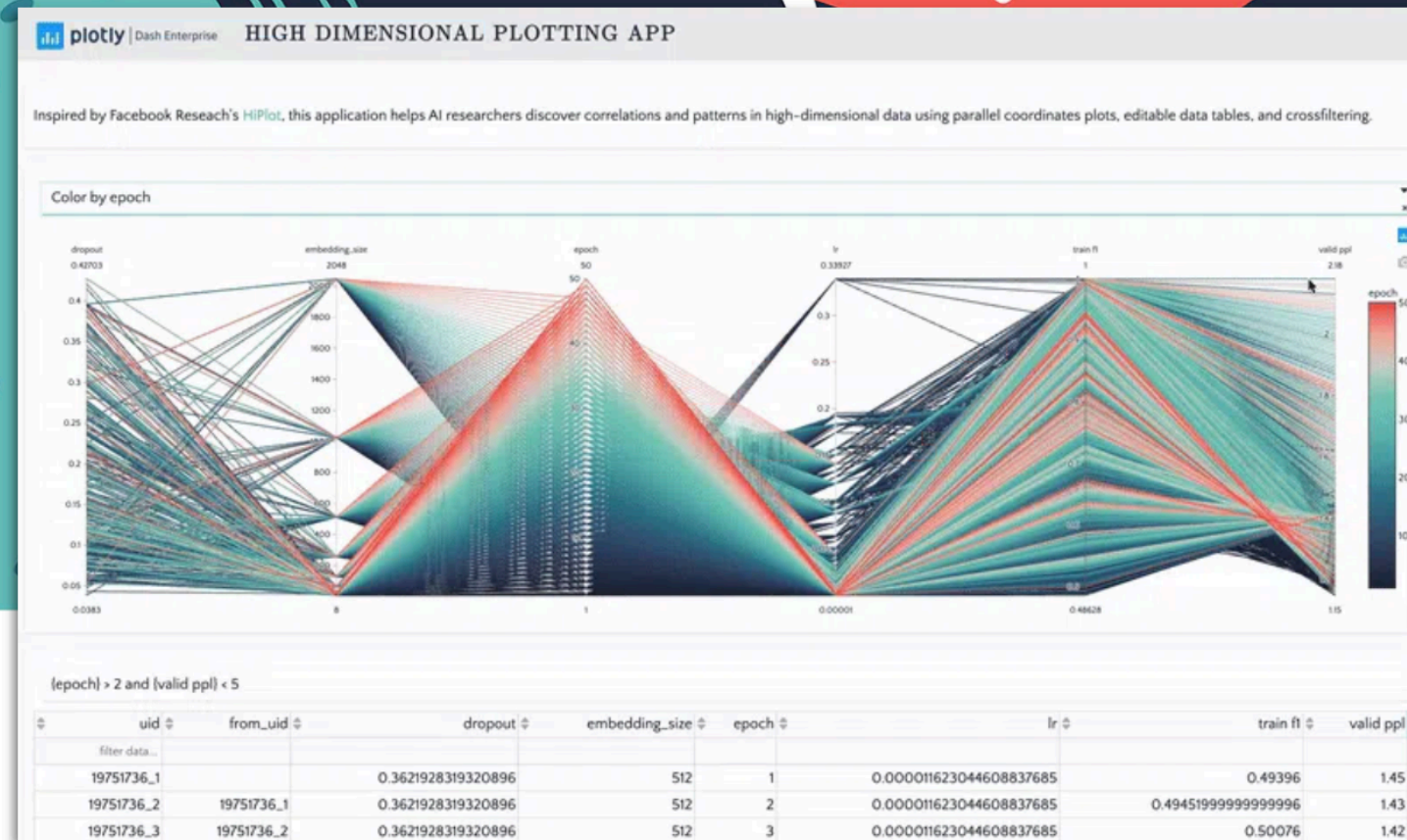
- Der Browser spricht nur JavaScript
- Nicht jeder mag JavaScript, aus Gründen

Mein Problem

- prototypische Applikation in R/Shiny
- Neu-Entwicklung der Anwendungslogik in Python
- ein Frontend muss her
- keine JavaScript-Affinität im Team

Die Lösung

~~Die Lösung~~
Eine Lösung



Build beautiful, web-based analytic apps. No JavaScript required.

The Dash platform empowers Data Science teams to focus on the data and models, while producing and sharing enterprise-ready analytic apps that sit on top of Python and R models. What would typically require a team of back-end developers, front-end developers, and IT can all be done with Dash.

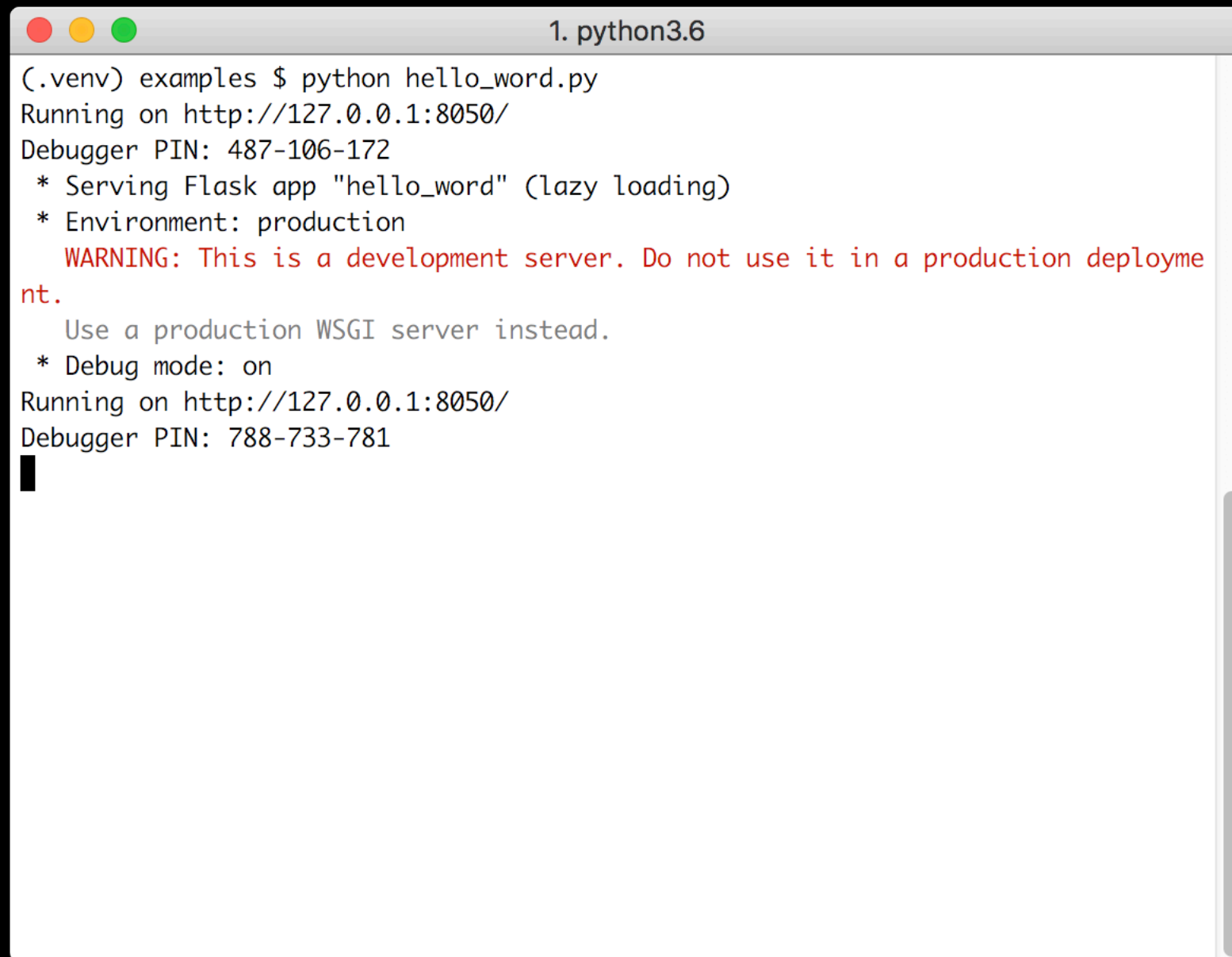
Dash

- Python-Framework für Web-Applikationen
- Bausteine
 - Layout
 - Komponenten
 - Callbacks
- MIT-Lizensiert

Die Basis

- Flask
- React.js
- Plotly.js
- zugrunde liegende Technologien komplett abstrahiert

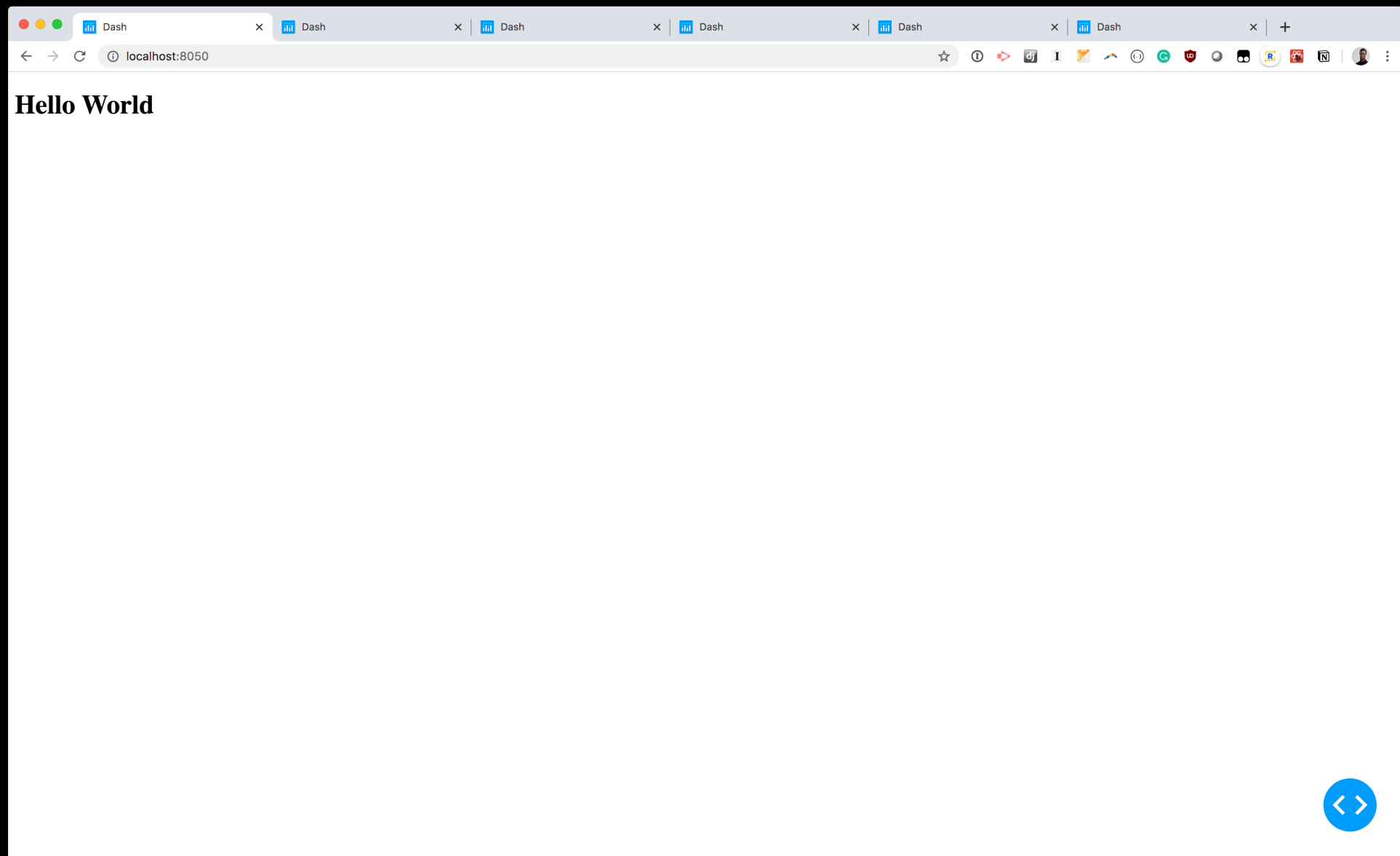
```
# -*- coding: utf-8 -*-  
import dash  
import dash_html_components as html  
  
app = dash.Dash(__name__)  
  
app.layout = html.Div(children=[  
    html.H1(children='Hello World'),  
])  
  
if __name__ == '__main__':  
    app.run_server(debug=True)
```

A terminal window titled "1. python3.6" with standard macOS window controls (red, yellow, green buttons). The terminal displays the output of running a Flask application. The text is as follows:

```
(.venv) examples $ python hello_word.py
Running on http://127.0.0.1:8050/
Debugger PIN: 487-106-172
* Serving Flask app "hello_word" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
Running on http://127.0.0.1:8050/
Debugger PIN: 788-733-781
█
```

Demo

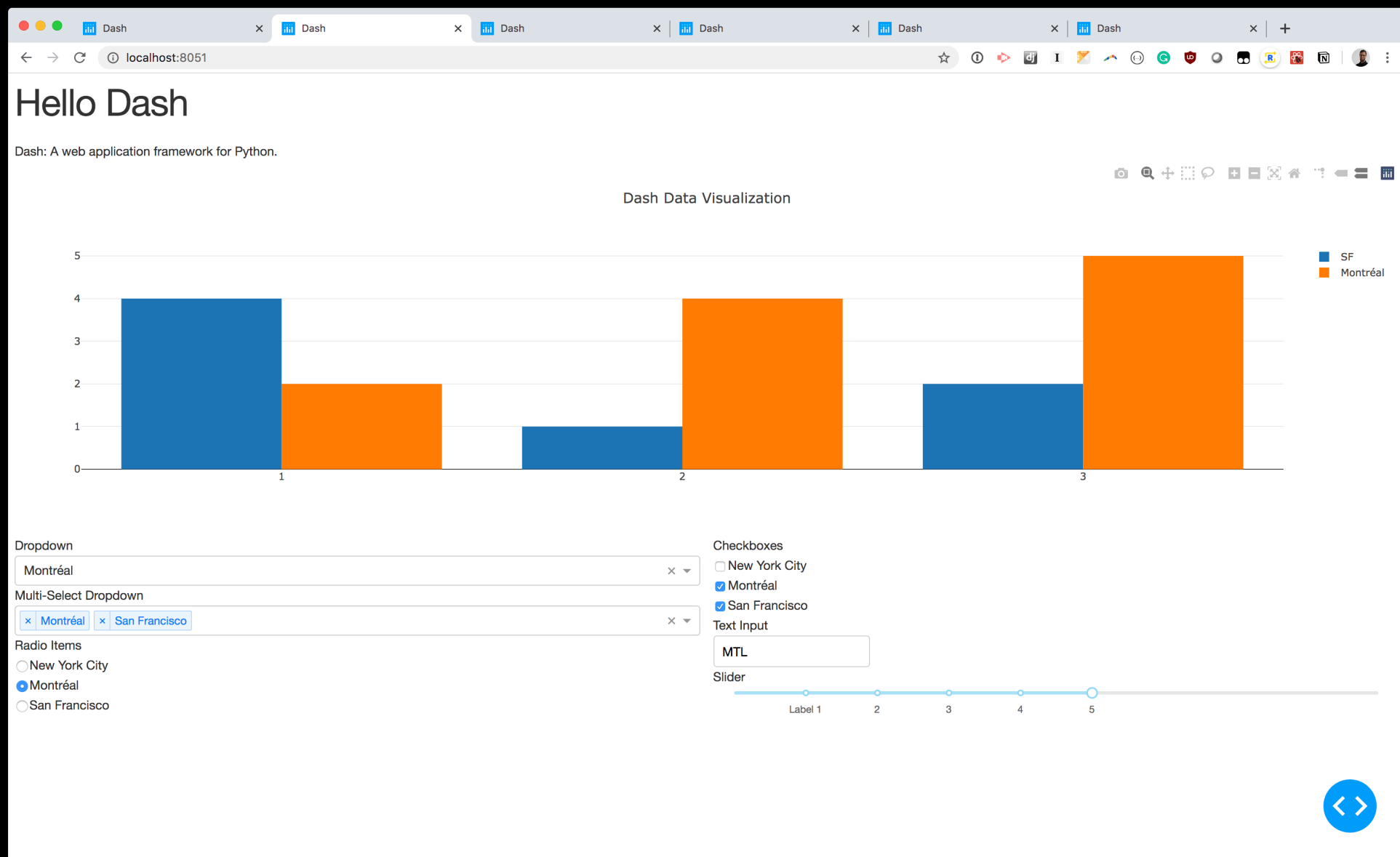
hello_world.py



Demo
hello_world.py

Dash Komponenten

- Core: Graph & Inputs
- HTML: HTML primitive
- DataTable: interaktive Tabellen-Komponente
- Bio: Komponenten für Bio-Informatik
- DAQ: komplexe Input-Komponenten
- Canvas: Image-Annotation und -Processing
- Cytoscape: Netzwerk-Visualisierung



Demo

hello_components.py

```
# -*- coding: utf-8 -*-
import dash
import dash_core_components as dcc
import dash_html_components as html

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for Python.
    '''),
    ...
])

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

```
dcc.Graph(  
    id='example-graph',  
    figure={  
        'data': [  
            {'x': [1, 2, 3], 'y': [4, 1, 2], 'type': 'bar', 'name': 'SF'},  
            {'x': [1, 2, 3], 'y': [2, 4, 5], 'type': 'bar', 'name': u'Montréal'},  
        ],  
        'layout': {  
            'title': 'Dash Data Visualization'  
        }  
    }  
) ,
```



```
html.Div([
    html.Label('Dropdown'),
    dcc.Dropdown(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': u'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value='MTL'
    ),

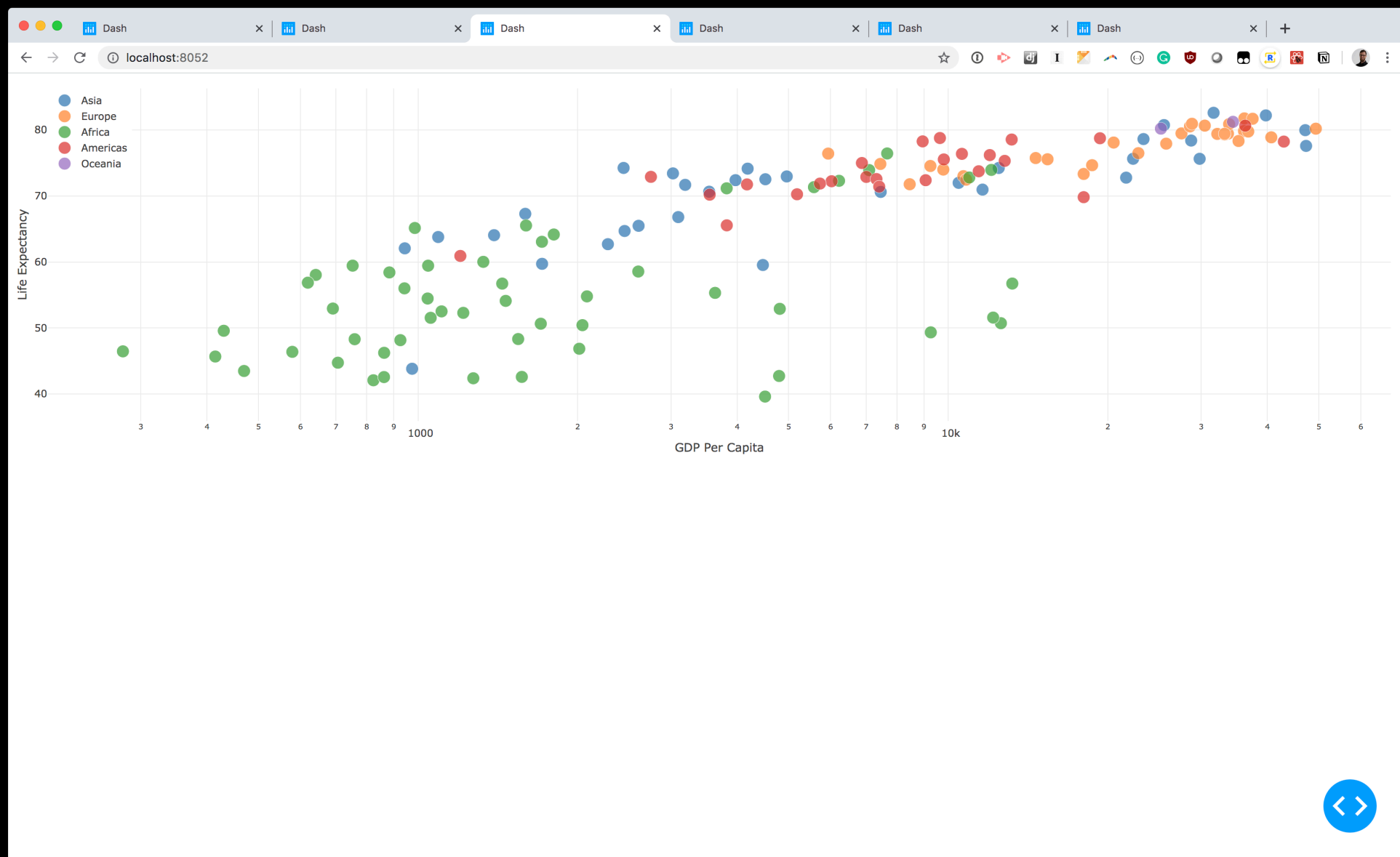
    html.Label('Multi-Select Dropdown'),
    dcc.Dropdown(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': u'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value=['MTL', 'SF'],
        multi=True
    ),

    html.Label('Radio Items'),
    dcc.RadioItems(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': u'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value='MTL'
    ),
```

```
html.Label('Checkboxes'),
dcc.Checklist(
    options=[
        {'label': 'New York City', 'value': 'NYC'},
        {'label': u'Montréal', 'value': 'MTL'},
        {'label': 'San Francisco', 'value': 'SF'}
    ],
    value=['MTL', 'SF']
),

html.Label('Text Input'),
dcc.Input(value='MTL', type='text'),

html.Label('Slider'),
dcc.Slider(
    min=0,
    max=9,
    marks={
        i: 'Label {}'.format(i) if i == 1 else str(i)
        for i in range(1, 6)
    },
    value=5,
),
], style={'columnCount': 2})
```



Demo

hello_graph.py

```

import dash
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

df = pd.read_csv('https://gist.githubusercontent.com/chriddyp/5d1ea79569ed194d432e56108a04d188/raw/a9f9e8076b837d541398e999dcbac2b2826a81f8/gdp-life-exp-2007.csv')

app.layout = html.Div([
    dcc.Graph(
        id='life-exp-vs-gdp',
        figure={
            'data': [
                dict(
                    x=df[df['continent'] == i]['gdp per capita'],
                    y=df[df['continent'] == i]['life expectancy'],
                    text=df[df['continent'] == i]['country'],
                    mode='markers',
                    opacity=0.7,
                    marker={
                        'size': 15,
                        'line': {'width': 0.5, 'color': 'white'}
                    },
                    name=i
                ) for i in df.continent.unique()
            ],
            'layout': dict(
                xaxis={'type': 'log', 'title': 'GDP Per Capita'},
                yaxis={'title': 'Life Expectancy'},
                margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
                legend={'x': 0, 'y': 1},
                hovermode='closest'
            )
        )
    )
])

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

```

Dash Callbacks

- Input
- Output
- State

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

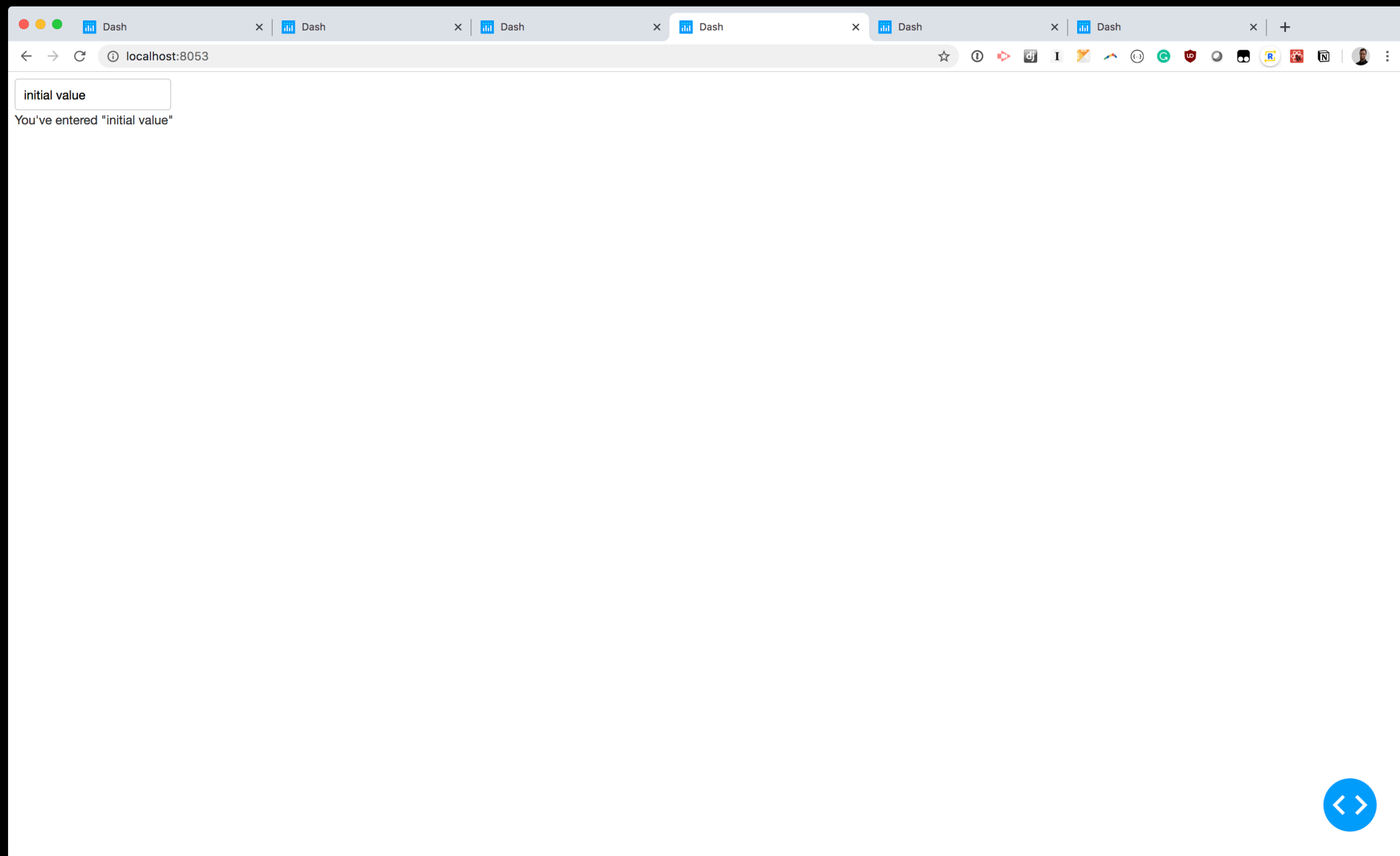
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    dcc.Input(id='my-id', value='initial value', type='text'),
    html.Div(id='my-div')
])

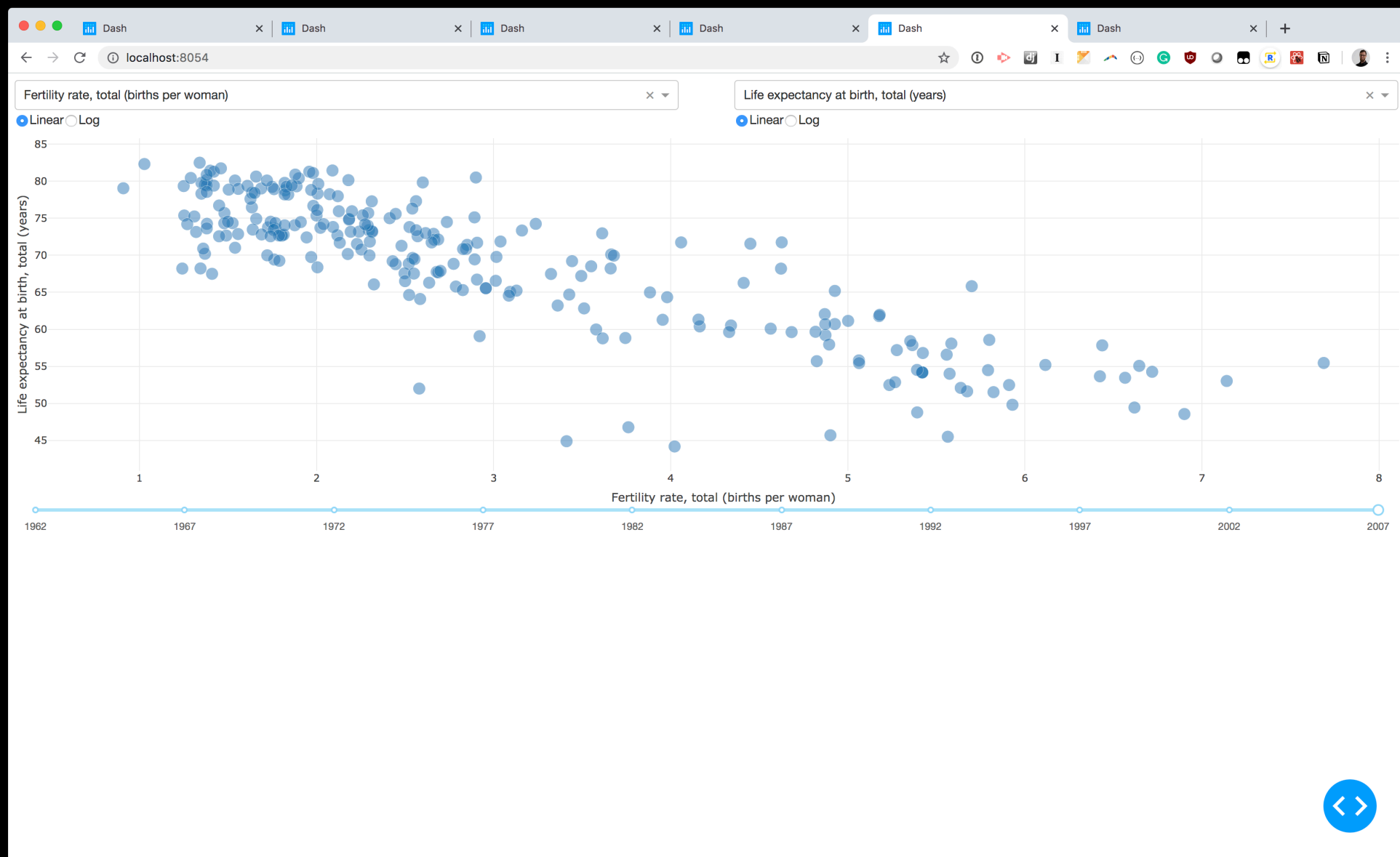
@app.callback(
    Output(component_id='my-div', component_property='children'),
    [Input(component_id='my-id', component_property='value')]
)
def update_output_div(input_value):
    return 'You\'ve entered "{}".format(input_value)

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



Demo

hello_callback.py



Demo

hello_interactive_graph.py


```

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

import pandas as pd

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

df = pd.read_csv('https://plotly.github.io/datasets/country_indicators.csv')

available_indicators = df['Indicator Name'].unique()

app.layout = html.Div([
    html.Div([
        html.Div([
            dcc.Dropdown(id='xaxis-column', ...),
            dcc.RadioItems(id='xaxis-type', ...)
        ], ...),
        html.Div([
            dcc.Dropdown(id='yaxis-column', ...),
            dcc.RadioItems(id='yaxis-type', ...)
        ], ...)
    ]),
    dcc.Graph(id='indicator-graphic'),
    dcc.Slider(id='year--slider', ...)
])

@app.callback(
    Output('indicator-graphic', 'figure'),
    [Input('xaxis-column', 'value'),
     Input('yaxis-column', 'value'),
     Input('xaxis-type', 'value'),
     Input('yaxis-type', 'value'),
     Input('year--slider', 'value')]
)
def update_graph(xaxis_column_name, yaxis_column_name, xaxis_type, yaxis_type, year_value):
    dff = df[df['Year'] == year_value]

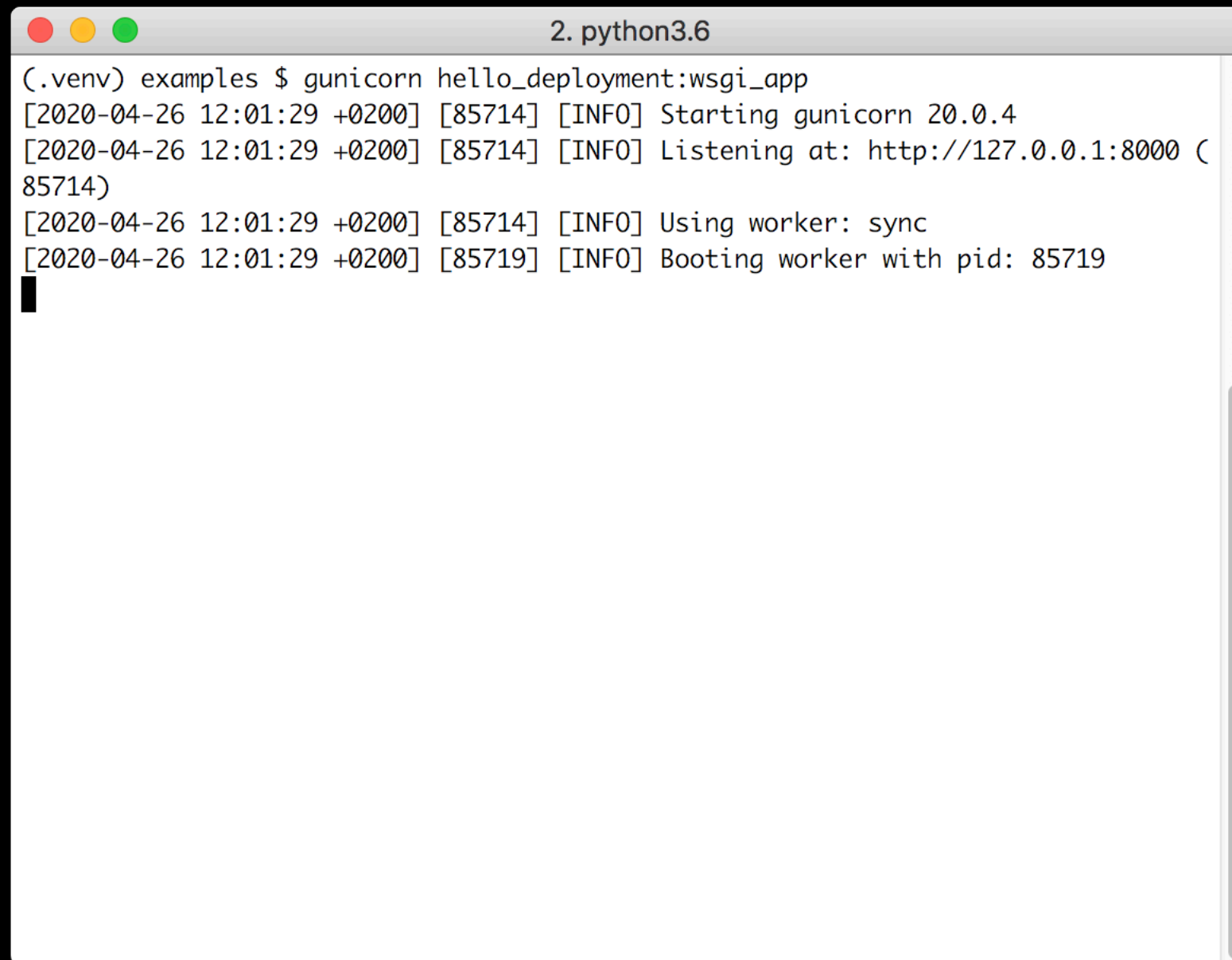
    return {
        'data': [...],
        'layout': dict(...)
    }

```

Deployment

- Dash-Apps sind Flask-Apps sind WSGI-Apps
- WSGI-Server wie gunicorn
- PaaS wie Heroku

```
# -*- coding: utf-8 -*-  
import dash  
import dash_html_components as html  
  
app = dash.Dash(__name__)  
  
app.layout = html.Div(children=[  
    html.H1(children='Hello WSGI'),  
)  
  
wsgi_app = app.server
```

A terminal window titled "2. python3.6" with standard macOS window controls (red, yellow, green buttons). The terminal displays the output of the command "gunicorn hello_deployment:wsgi_app". The logs show the start of gunicorn 20.0.4, listening on http://127.0.0.1:8000 (85714), using the sync worker, and booting a worker with pid 85719. A cursor is visible on the line following the last log entry.

```
(.venv) examples $ gunicorn hello_deployment:wsgi_app
[2020-04-26 12:01:29 +0200] [85714] [INFO] Starting gunicorn 20.0.4
[2020-04-26 12:01:29 +0200] [85714] [INFO] Listening at: http://127.0.0.1:8000 (
85714)
[2020-04-26 12:01:29 +0200] [85714] [INFO] Using worker: sync
[2020-04-26 12:01:29 +0200] [85719] [INFO] Booting worker with pid: 85719
█
```

Demo

hello_deployment.py

Fallstricke & Nachteile

- jeder* Callback löst einen HTTP-Request aus
- Applikations-Server notwendig
- Umdenken für Web-Entwickler
- Interessenskonflikt (?) mit Dash Enterprise
- Keine Server-side push out of the box
- Es bleibt JavaScript

Alternativen

- brython
- pyjs
- Anvil (€€€)
- in den sauren Apfel beißen

EOF

- Slides und Code-Beispiele:
<https://github.com/consideratecode/dash-pycamp2020>
- @danielhepper