Adatbányászat a Gyakorlatban

6. Gyakorlat: Haladó Dash módszerek

Kuknyó Dániel
Budapesti Gazdasági Egyetem

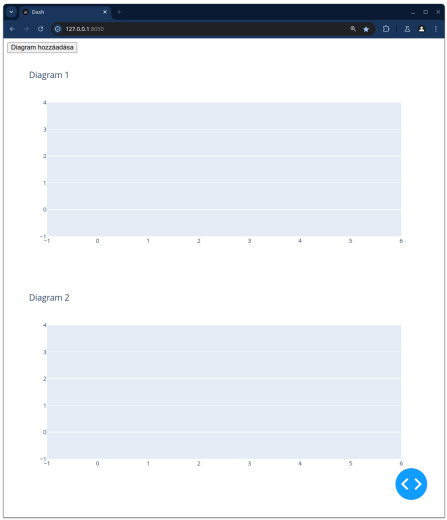
2024/25
1.félév

Dinamikus felhasználói komponensek (dyn_component_app_v1.py)

Olyan dinamikus komponensek, amelyek nem állandóak, hanem felhasználói interakcióra kerülnek hozzáadásra az alkalmazáshoz, és el is lehet őket távolítani.

Ehhez tartozóan az első alkalmazás elrendezése:

```
1 app.layout = html.Div([
2   dbc.Button("Diagram hozzáadása", id='
        dyn_component_button'),
3   html.Div(id='dyn_component_output',
        children=[]),
4 ])
```



Callback függvény felhasználói komponensek hozzáadására

A callback a `dyn_component_output` `Div` `children` komponensét frissíti.

Ha a gombot megnyomták, létrehoz egy új oszlopdiagramot, és a diagram címében megjeleníti a gombnyomások számát. Az új diagramot hozzáadja a `children` listához.

Visszatér a frissített `children` listával, amely tartalmazza az új diagramot.

```
@app.callback(
  Output('dyn_component_output', '
      children'),
  Input('dyn_component_button', '
      n_clicks'),
  State('dyn_component_output', '
      children')
)
def add_new_chart(n_clicks, children):
  if not n_clicks:
    return no_update
  new_chart = dcc.Graph(figure=px.bar(
      title=f"Diagram {n_clicks}"))
  children.append(new_chart)
  return children
```

Az alkalmazás callback gráfja

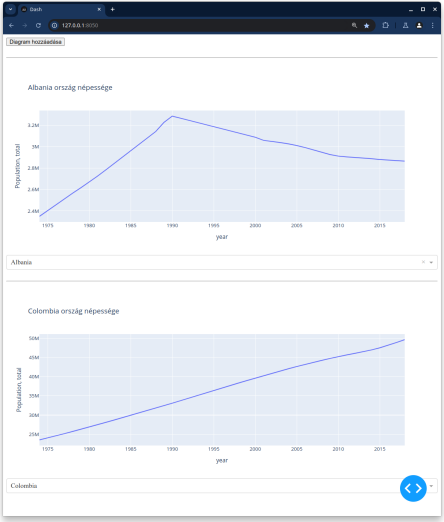
A callback gráf ebben az esetben egy speciális hurkot mutat. Ez azt jelenti, hogy a függvény egy olyan komponenst térít vissza, amit megkapott paraméterül és módosított a függvénytörzsben.

Ez a diagram változatlan marad a hozzáadott komponensek számától függetlenül.

Mintaillesztő callback függvények

Az alkalmazás következő verziója interaktivitást ad a felhasználói komponenseknek, mintaillesztő callback függvények segítségével.

Minden diagramnak van egy saját legördülő menüje, ahol ki lehet választani egy országnevet, majd kirajzolja a népességet a diagramra.



Dinamikus komponens elnevezések

Dash rendszerben egy komponens `id` paramétere bármilyen hash-képes (egy függvényen keresztül egyértelműen leképezhető) objektum lehet, így pl. egy szótár is.

Az alkalmazásban a `id` attribútum egy szótár, amely tartalmazza a `type` és `index` kulcsokat. Ez a struktúra lehetővé teszi komponenseket dinamikus azonosítását az alkalmazásban. Például több diagram létrehozása és módosítása esetén mindegyiket egyedi névvel látja el.

```
# Új diagram létrehozása
new_chart = dcc.Graph(
  id={'type': 'chart', 'index':
      n_clicks},
  figure=px.bar(title=f"Diagram {
      n_clicks}")
)
# Legördülő lista opciók létrehozása
countries = poverty[poverty['is_country
    ']]['Country Name'].drop_duplicates
    ().sort_values()
# Legördülő lista létrehozása
new_dropdown = dcc.Dropdown(
  id={'type': 'dropdown', 'index':
      n_clicks},
  options=[{'label': c, 'value': c} for
       c in countries],
  placeholder='Ország kiválasztása'
)
```

Dash MATCH

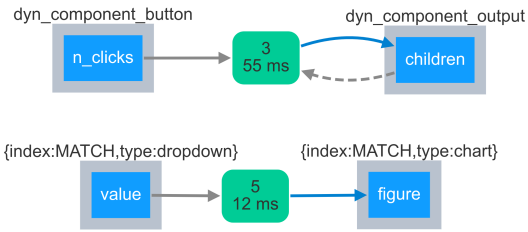
A `dash.dependencies.MATCH` lehetővé teszi, hoyg a callback egy adott komponenscsoport egy adott példányára vonatkozzon.

Például ha több dinamikus objektum közül mindegyiknek van egy egyedi `id` paramétere egy szótár formájában, akkor a `MATCH` segítségével meg lehet határozni, hogy a callback csak az adott indexnek megfelelő komponensre reagáljon.

```
@app.callback(
  Output({'type': 'chart', 'index':
      MATCH}, 'figure'),
  Input({'type': 'dropdown', 'index':
      MATCH}, 'value'),
)
def create_population_chart(country):
  if not country:
    return no_update
  # Adatkészlet szűrése
  df = poverty[poverty['Country Name']
      == country]
  # Diagram létrehozása
  fig = px.line(
    df,
    x='year',
    y='Population, total',
    title=f'{country} ország népessége'
  )
  return fig
```

Az alkalmazás callback gráfja

A felső gráf megegyezik az alkalmazás előző verziójában látottakkal. Az alsó gráfon a `MATCH` jelzi, hogy csak a megfelelő indexű komponenseket frissítse.



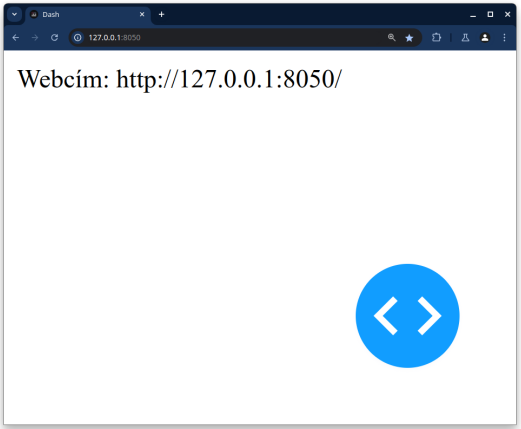
A `MATCH` mellett az `ALL` használható minden komponens nevének illesztésére, az `ALLSMALLER` pedig azokra a komponensekre, amelyek a callback függvény indulása előtt lettek létrehozva.

Location és Link komponensek (multi_app_v1.py)

Dash keretrendszerben a Location és Link komponensek az útválasztás fontos elemei, melyek lehetővé teszik az URL kezelést és a navigációt.

Location

A Dash alkalmazásban történő kezelésére szolgál. A Location komponens adatai közvetlenül a böngésző URL-jéből származnak, és az alkalmazásban történő változásokat közvetlenül a böngésző URL-jébe írja vissza.



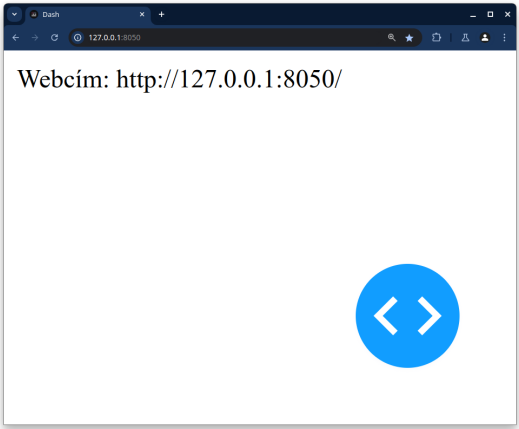
Location és Link komponensek (multi_app_v1.py)

Dash keretrendszerben a `Location` és `Link` komponensek az útválasztás fontos elemei, melyek lehetővé teszik az URL kezelést és a navigációt.

Link

A `Link` komponens egy hiperhivatkozást hoz létre a Dash alkalmazásban. Fontosabb attribútumai:

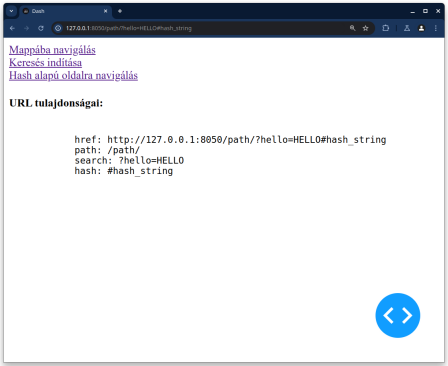
- `href`: Meghatározza a hivatkozás URL-jét
- `target`: Megadja, hogy a hivatkozást az aktuális vagy új ablakban kell-e megnyitni
- `refresh`: Ha igaz, akkor az URL megnyitásakor frissül az oldal



Location és Link komponensek (multi_app_v2.py)

Az alkalmazás a Link komponens felhasználásával megváltoztatja az URL-t és a Location komponens segítségével pedig kivonatol tetszőleges attribútumokat.

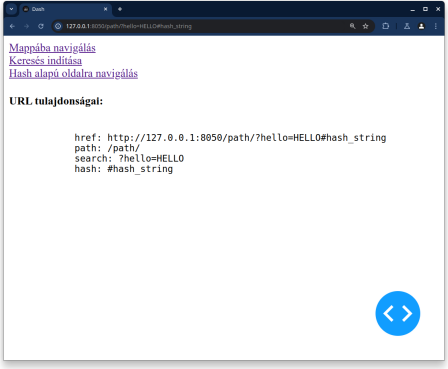
```
1 app.layout = html.Div([
2   dcc.Location(id='location', refresh=False),
3   html.A(href='/path', children='Mappába navig
        álás'),
4   dcc.Link(href='/path/search?one=1&two=2',
        children='Keresés indítása'),
5   dcc.Link(href='/path/?hello=HELLO#
        hash_string', children='Hash alapú
        oldalra navigálás'),
6   html.H4("URL tulajdonságai:"),
7   html.Div(id='location_output')
8 ])
```



Location és Link komponensek (multi_app_v2.py)

A callback függvény bemenetében a `Location` több paramétere található, és a formázást megőrző `html.Pre` majd kiíratódnak a `Div` tárolóra.

```
@app.callback(
  Output('location_output', 'children'),
  Input('location', 'pathname'),
  Input('location', 'search'),
  Input('location', 'href'),
  Input('location', 'hash'),
)
def show_url(pathname, search, href, hash):
  return html.Div([
    html.Pre([
      f"""href: {href}
      path: {pathname}
      search: {search}
      hash: {hash}"""
    ])
  ])
```



Többlapos alkalmazások struktúrája

Többlapos alkalmazások esetén az elrendezés csontvázát egy fő elrendezés komponens adja, amelyben egy üres `Div` található. A `Div` tartalma attól függően fog változni, hogy a felhasználó milyen oldalra navigál az alkalmazáson belül.

1. Importok (boilerplate):

```
import dash
from dash import dcc
...
```

2. Alkalmazás példányosítása:

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

3. Alkalmazás elrendezése:

```
app.layout = html.Div([
  ...
])
```

4. Callback függvények:

```
@app.callback()
  ...
@app.callback()
  ...
```

5. Alkalmazás futtatása:

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

Többlapos alkalmazás: fő elrendezés

Ez a komponens szolgál az alkalmazás csontvázaként.

Tartalmaz egy `NavbarSimple` navigációs sávot és az országokat tartalmazó legördülő menüt. Az elrendezés még magában foglalja a lapos elrendezést az oldal alján.

A lap törzse tartalmaz egy `Location` komponenst és egy üres `Div` objektumot, ami majd az elrendezést fogja tartalmazni.

```
main_layout = html.Div([
  dbc.NavbarSimple([
    ...
  ]),
  dcc.Location(id='location'),
  html.Div(
    id='main_content',
    children=[...],
  ),
  dbc.Tabs([
    ...
  ]),
])

...

app.layout = main_layout
```

Többlapos alkalmazás: Indikátorok műszerfala

Ez az az elrendezés komponens, ami az eddigi alkalmazásokban került fejlesztésre. Azzal a különbséggel, hogy egy új változóba kerül elmentésre, és átadódik a `main_content` `Div` komponensnek, ha a megfelelő feltételek teljesülnek (az URL nem tartalmaz egy országnevet).

```
1 indicators_dashboard = html.Div([
2   # Minden eddigi komponens
3 ])
```

Többlapos alkalmazás: Országok műszerfala

Ez a komponens is egy külön változóba kerül elmentésre, és akkor jelenítődik meg, amikor az URL tartalmaz egy országnevet. Ebben a komponensben az elrendezés (grafikon és táblázat) az URL-ben szereplő országnévtől függően változhat.

```
1 country_dashboard = html.Div([
2   html.H1(id='country_heading'),
3   dcc.Graph(id='country_page_graph'),
4   dcc.Dropdown(id='
        country_page_indicator_dropdown')
        ,
5   dcc.Dropdown(id='
        country_page_contry_dropdown'),
6   html.Div(id='country_table')
7 ])
```

Többlapos alkalmazás: Validációs elrendezés

Ez a komponens egy egyszerű lista egy Dash komponens formájában, ami az előző három elrendezést tartalmazza.

Ennek a jelentősége, hogy megadja az alkalmazás számára, hogy milyen elrendezések érhetőek el a számára. Amikor az alkalmazás csak egy elrendezésből jelenít meg komponenseket és a többiből nem, némely komponensek nem lesznek az alkalmazás része, és néhány callback függvény eltörik.

A validációs elrendezés egy egyszerű módszert ad az alkalmazás egésze számára elérhető elrendezések kezelésére.

```
app.validation_layout = html.Div([
  main_layout,
  indicators_dashboard,
  country_dashboard,
])
```

Tartalom megjelenítése az URL-től függően

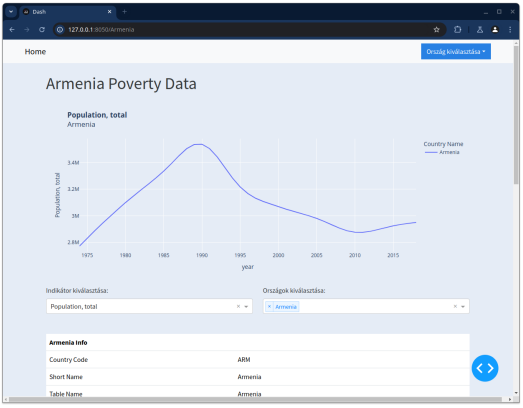
A következő callback függvény ellenőrzi, hogy a `Location` komponens egyike-e az elérhető országoknak vagy sem. Ha igen, visszatéríti a `country_dashboard` elrendezést, egyébként pedig az `indicators_layout` elrendezést.

```
countries = poverty[poverty['is_country
    ']]['Country Name'].drop_duplicates
    ().sort_values().tolist()
...
@app.callback(
  Output('main_content', 'children'),
  Input('location', 'pathname')
)
def display_content(pathname):
if unquote(pathname[1:]) in countries:
  return country_dashboard
else:ar
  return indicators_dashboard
```

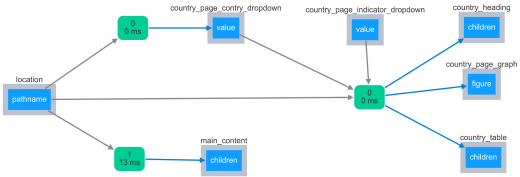
Alkalmazás többlapos támogatással (app_v6_1.py)

Az alkalmazás következő verziójához az összes eddig implementált funkcionalitást be kell építeni a korábban létrehozott struktúrába.

Ennek eredménye egy konstans navigációs sáv, amelyben kiválasztható a megfelelő ország. A sávon kiválasztható az ország, amely átnavigálja a felhasználót egy új oldalra, amihez tartozóan az országhoz tartozó specifikus adatok fognak megjelenni.



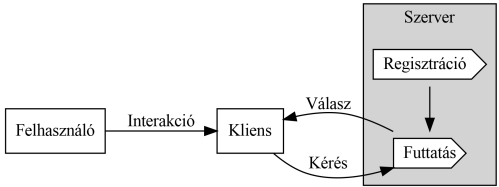
Az alkalmazás callback gráfja



Szerveroldali callback függvények

A szerveroldali függvények regisztrációjának és futtatásának folyamata a következőképpen írható le:

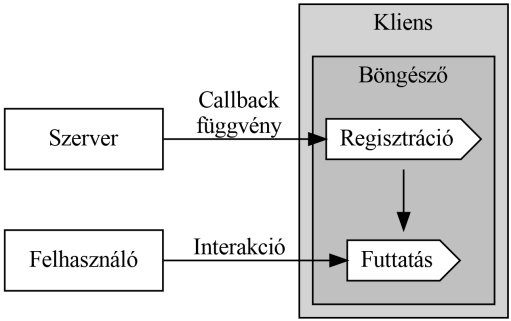
1. Függvény definiálása
2. Szerver regisztrálja a callback függvényt
3. Felhasználói interakció
4. Callback futtatása a szerveren
5. Eredmény visszaküldése a kliensnek



Kliensoldali callback függvények

A kliensoldali függvények folyamata:

1. Függvény definiálása
2. Szerver elküldi a függvényt a kliens oldalra
3. Regisztráció a kliens oldalon
4. Felhasználói interakció
5. Callback futtatása a böngészőben



Kliensoldali callback függvények

A következő két programrészlet ugyanazt a callback függvényt mutatja be szerver- és kliens oldalon.

```python
from dash import Input, Output

@callback(
  Output('out-component', 'value'),
  Input('in-component1', 'value'),
  Input('in-component2', 'value')
)
def large_params_function(largeValue1,
    largeValue2):
  largeValueOutput = someTransform(
      largeValue1, largeValue2)
return largeValueOutput
```

```python
from dash import clientside_callback,
    Input, Output

clientside_callback(
  """
  function(largeValue1, largeValue2) {
    return someTransform(largeValue1,
        largeValue2);
  }
  """,
  Output('out-component', 'value'),
  Input('in-component1', 'value'),
  Input('in-component2', 'value')
)
```