Tableaux de bord avec Python Dash

1. Première approche

Salim Lardjane
Université de Bretagne Sud

Introduction

- Dash est un cadre de travail open source créé par l'équipe de développement de plotly qui est basé sur Flask, plotly.js et react.js et qui permet de construire des tableaux de bord interactifs.

- La première version de Dash date de juin 2017.

- Sa caractéristique principale est qu'il permet de développer des applications Web interactives de visualisation entièrement en Python.

Introduction

- Un autre avantage de Dash est qu'en utilisant Python, on peut avoir recours à Pandas et à d'autres librairies dédiées pour manipuler les données.

- De plus, on a accès à toutes les fonctionnalités graphiques de plotly (https://plot.ly).

Installation

- Pour installer Dash sous Anaconda, il suffit de soumettre la commande suivante dans une fenêtre de terminal :

```
conda install -c conda-forge dash
```

Premier exemple

- Pour tester que tout fonctionne, on soumet le programme suivant :

PYTHON

```
import dash
import dash_core_components as dcc
import dash_html_components as html
```

Premier exemple

```
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.
    '''),

    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'
            }
        }
    )
])
```

Premier exemple

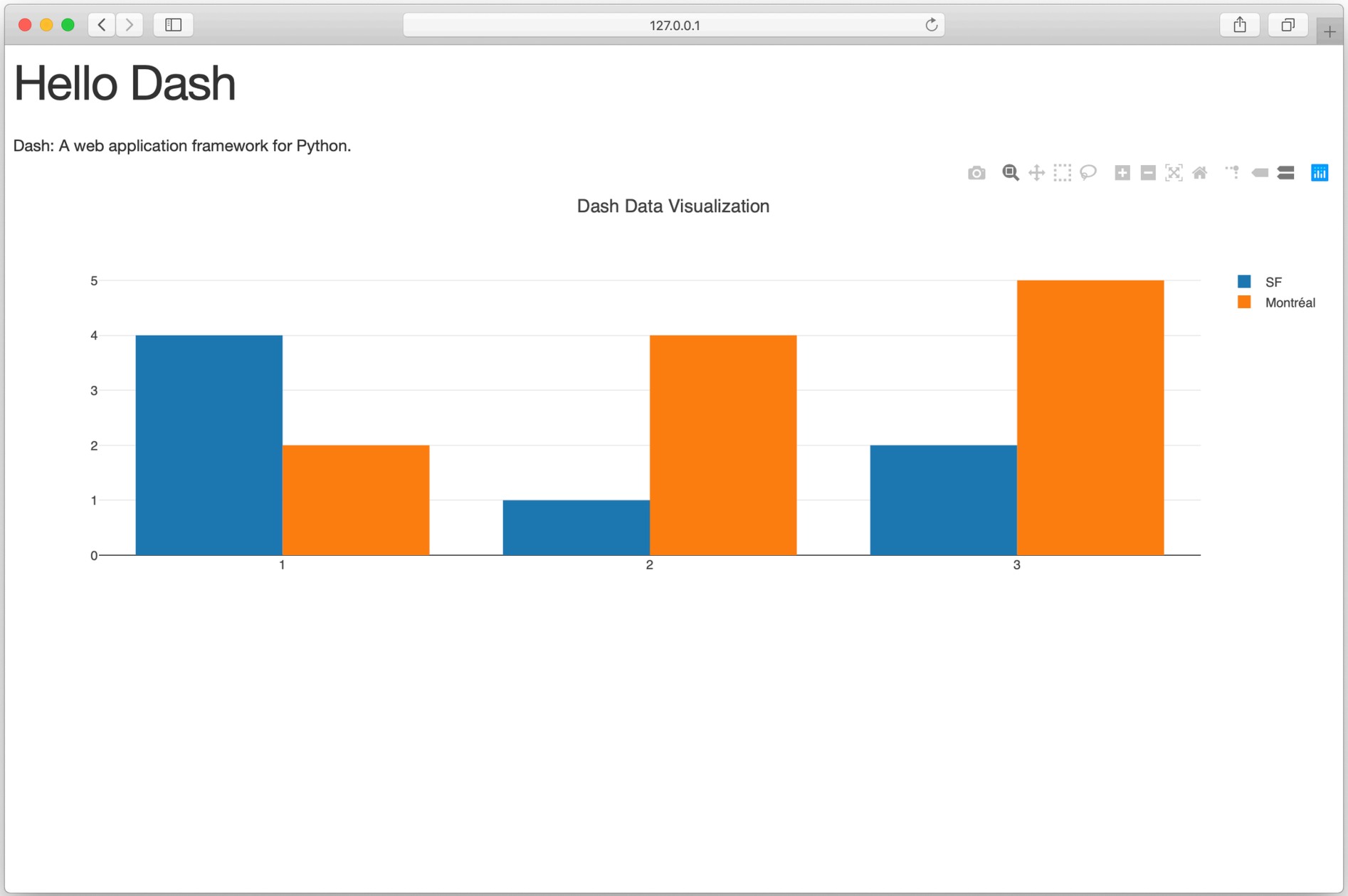
```
if __name__ == '__main__':
    app.run_server(debug=True)
```

- On exécute ensuite le programme, ce qui doit afficher dans la console un message du type :

… **Running on http://127.0.0.1:8050** …

- On peut alors consulter l'application Web générée à l'aide d'un navigateur. Sur ce premier exemple, on obtient :

Premier exemple



Premier exemple

- Notons que les applications Dash sont toujours composées de deux parties :

- La première partie est le « layout » de l'application, qui contrôle son apparence.

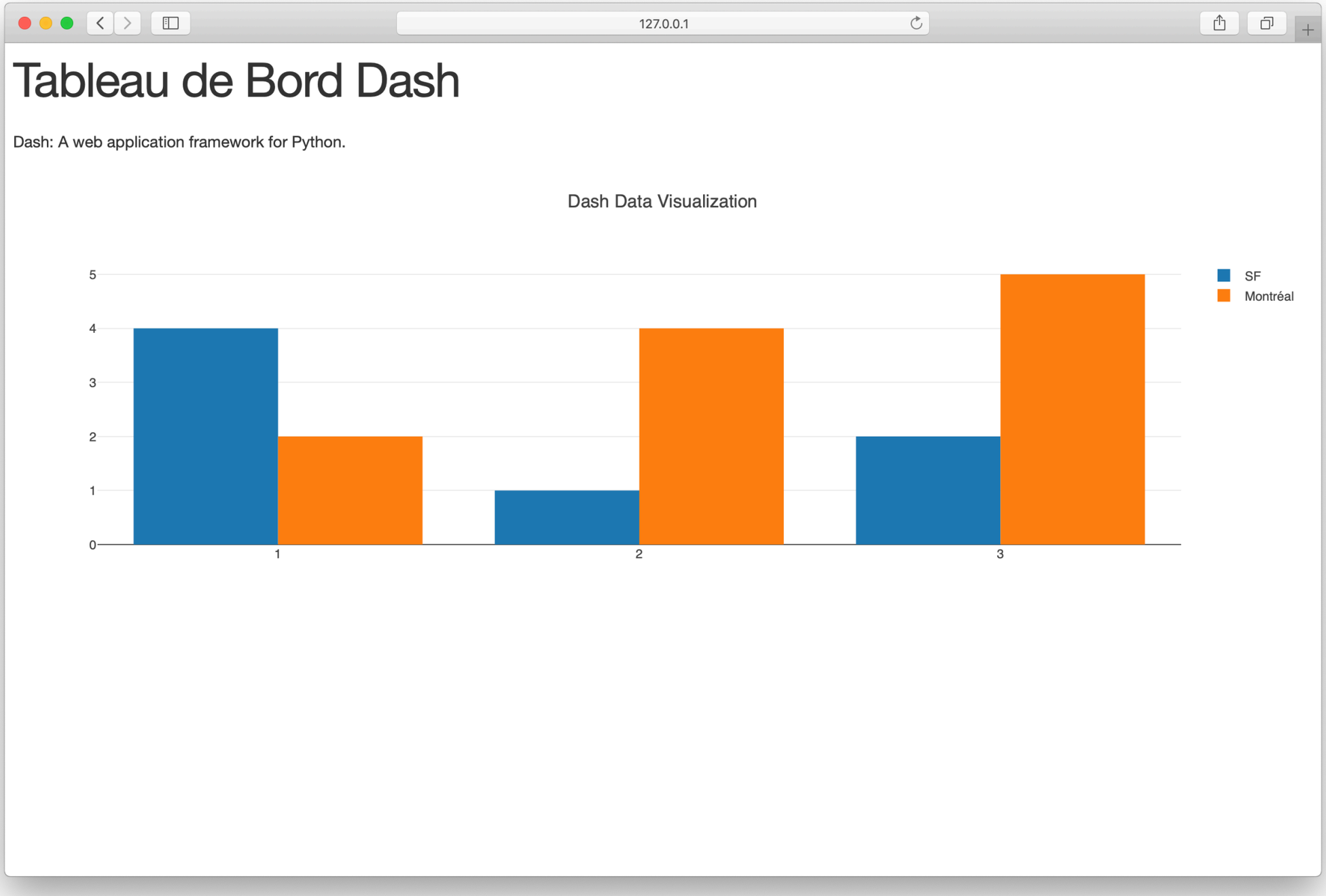
- La deuxième partie contrôle l'interactivité de l'application. Nous la verrons plus en détail dans la suite.

Premier exemple

- Dash implémente le « hot-reloading », c'est-à-dire que si l'on modifie le code et qu'on le sauvegarde, l'application est automatiquement redémarrée et le changement apparait aussitôt dans la fenêtre du navigateur.

- Pour désactiver cette fonctionnalité, on peut faire : `app.run_server(dev_tools_hot_reload=False)`.

Premier exemple



Premier exemple

- La librairie dash_html_components contient une classe dédiée pour chaque balise HTML, avec des arguments correspondant aux argument HTML.

- Modifions par exemple le style des éléments HTML dans le programme précédent :

Premier exemple

PYTHON

```
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)

colors = {
    'background': '#111111',
    'text': '#7FDBFF'
}
```

Premier exemple

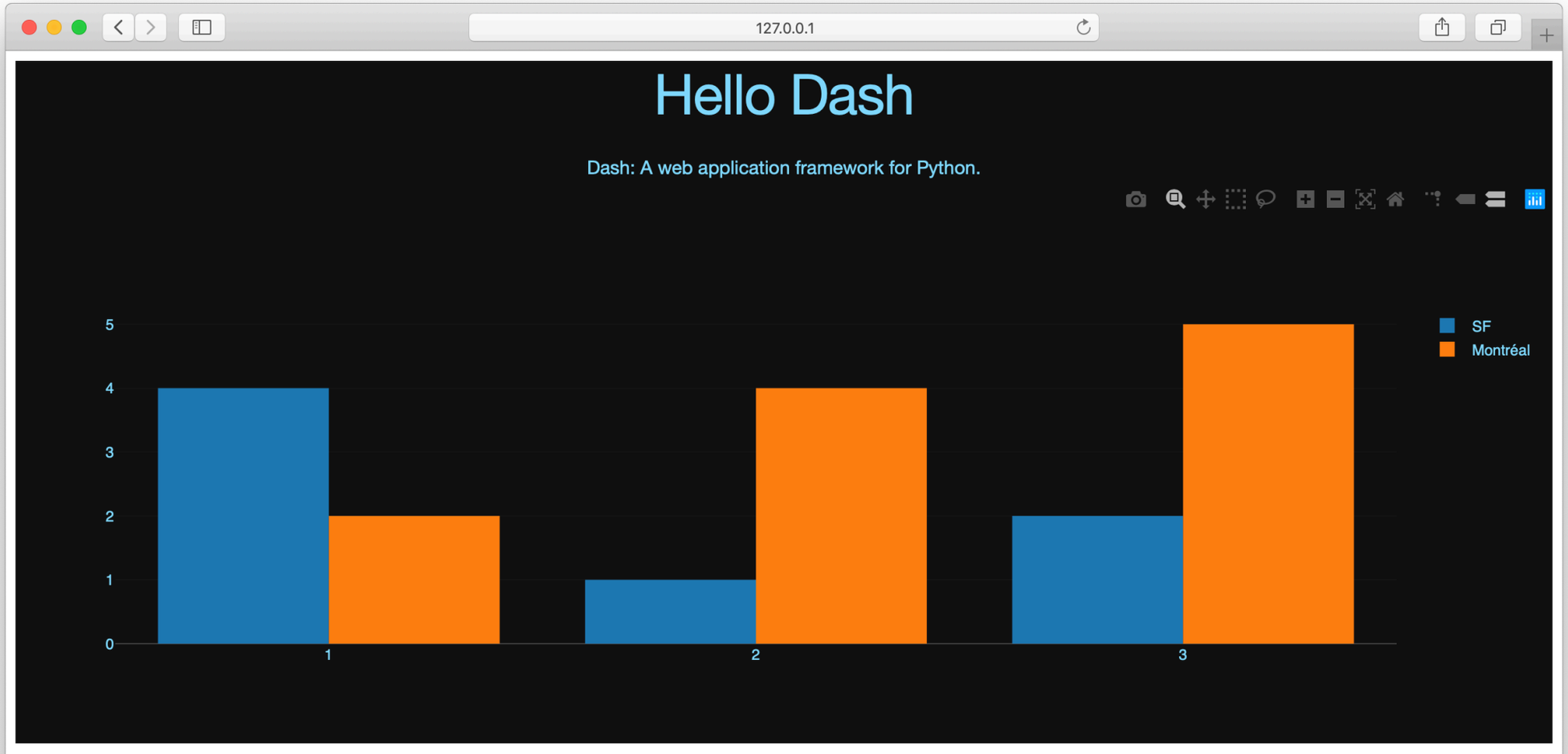
```
app.layout = html.Div(style={'backgroundColor': colors['background']}, children=[
    html.H1(
        children='Hello Dash',
        style={
            'textAlign': 'center',
            'color': colors['text']
        }
    ),

    html.Div(children='Dash: A web application framework for Python.', style={
        'textAlign': 'center',
        'color': colors['text']
    }),

    dcc.Graph(
        id='example-graph-2',
        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': {
                'plot_bgcolor': colors['background'],
                'paper_bgcolor': colors['background'],
                'font': {
                    'color': colors['text']
                }
            }
        }
    )
])
```

Premier exemple

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

- A la sauvegarde du programme, on obtient dans le navigateur, après rafraîchissement (sous Safari) :

Premier exemple



Premier exemple

- Dans le programme précédent, on a modifié le style des éléments **`HTML.Div`** et **`HTML.H1`**.

```
html.H1('Hello Dash', style={'textAlign': 'center', 'color': ‘#7FDBFF’})
```

est rendu par Dash pour la navigateur comme :

```
<h1 style="text-align: center; color: #7FDBFF">Hello Dash</h1>
```

Premier exemple

- Les différences entre les attributs `dash_html_components` et les attributs HTML sont les suivantes :

- La propriété style de HTML est une chaîne de caractères de séparateur point-virgule. En Dash, il suffit de fournir un dictionnaire.

- Les clefs du dictionnaire style sont camelCased. Ainsi, au lieu de text-align, c'est textAlign.

Premier exemple

- L’attribut class de HTML est traduit par className en Dash.

- Les descendants d’une balise HTML sont spécifiés par le mot-clef chidren. En Dash, il correspond souvent au premier argument et le mot-clef est donc omis.

- *Ces différences mises à part, l’ensemble des attributs et balises HTML sont disponibles sous Dash.*

Éléments réutilisables

- Sous Dash, on peut programmer des éléments réutilisables partout dans le code, comme des tables.

- L’exemple suivant génère une « table » à partir d’un tableau de données Pandas :

Éléments réutilisables

PYTHON

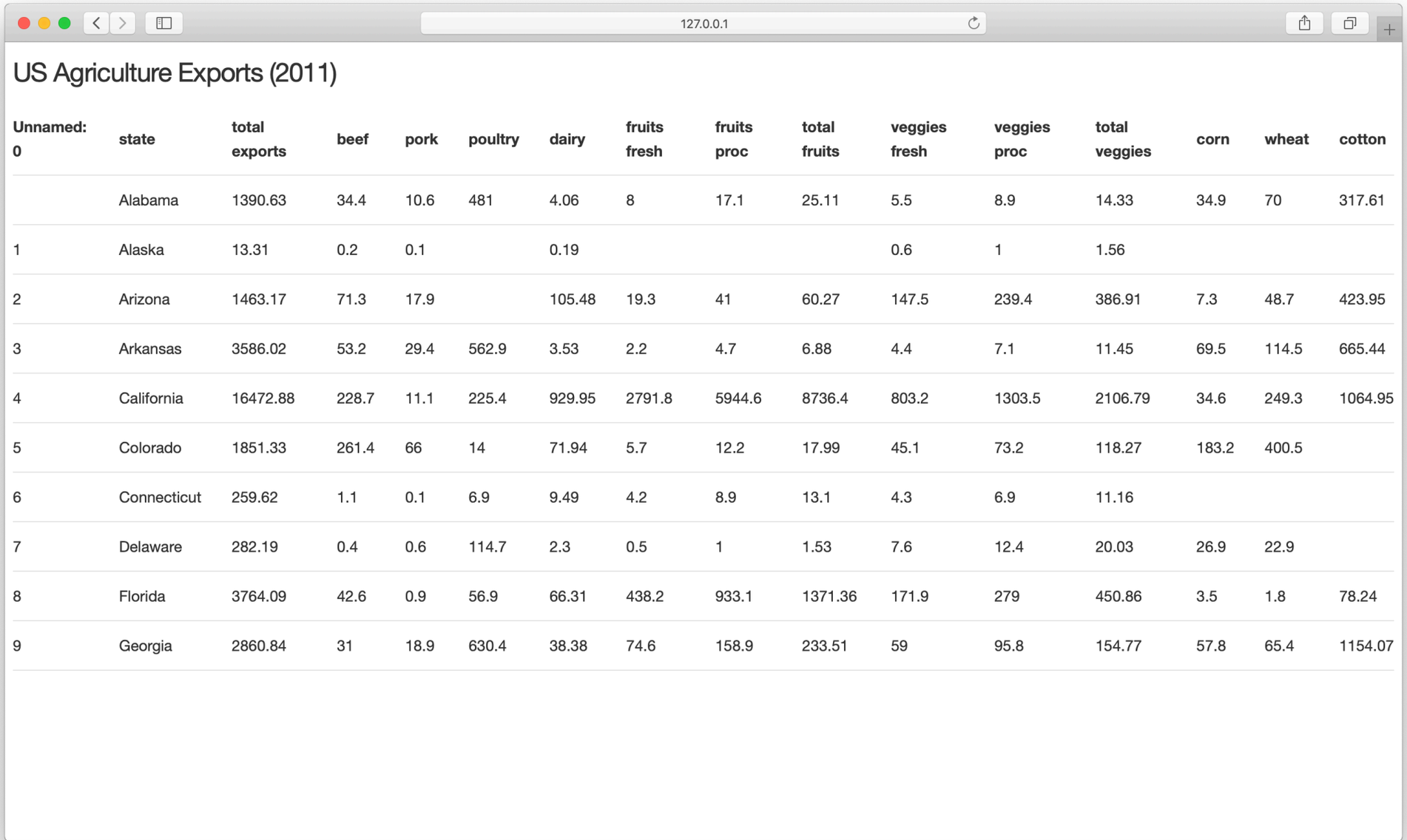
```
import dash
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd

df = pd.read_csv(
    'https://gist.githubusercontent.com/chriddyp/'
    'c78bf172206ce24f77d6363a2d754b59/raw/'
    'c353e8ef842413cae56ae3920b8fd78468aa4cb2/'
    'usa-agricultural-exports-2011.csv')


def generate_table(dataframe, max_rows=10):
    return html.Table(
        # Header
        [html.Tr([html.Th(col) for col in dataframe.columns])] +

        # Body
        [html.Tr([
            html.Td(dataframe.iloc[i][col]) for col in dataframe.columns
        ]) for i in range(min(len(dataframe), max_rows))]
    )
```

Éléments réutilisables

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

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

app.layout = html.Div(children=[
    html.H4(children='US Agriculture Exports (2011)'),
    generate_table(df)
])

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

- Le résultat est le suivant :

Éléments réutilisables



Visualisation

- La librairie dash_core_components a un élément nommé Graph.

- Graph permet d'obtenir des visualisations interactives à partir de la librairie javascript open source de visualisation plotly.js.

- Plotly.js implémente plus de 35 types de graphiques différents (selon les versions) et permet d'obtenir les graphiques au formats SVG vectoriel et WebGL.

Visualisation

- L’exemple suivant permet d’obtenir un nuage de points (ang. scatterplot) à partir d’un tableau de données Pandas :

Visualisation

PYTHON

```
import dash
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd
import plotly.graph_objs as go

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')
```

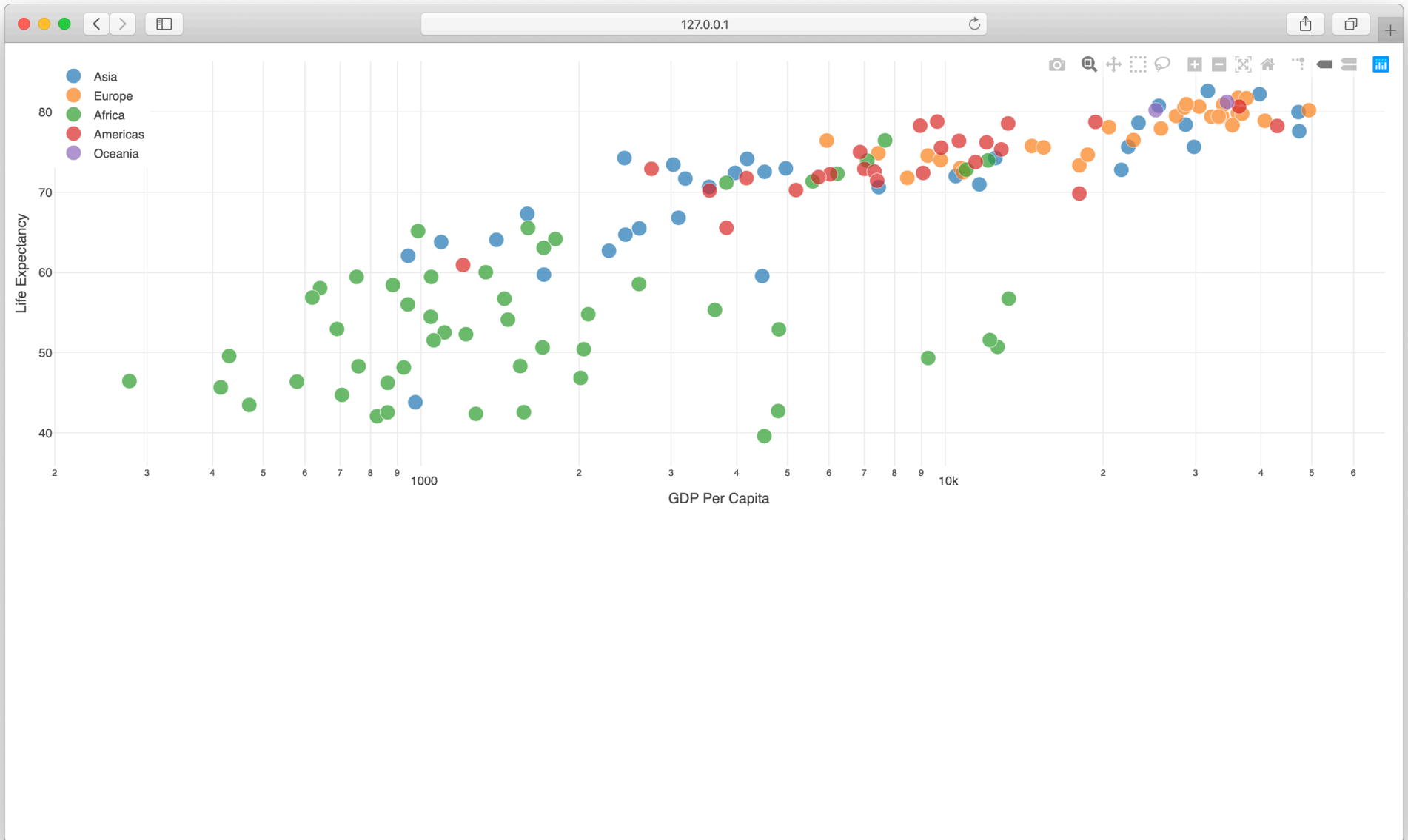
Visualisation

```
app.layout = html.Div([
    dcc.Graph(
        id='life-exp-vs-gdp',
        figure={
            'data': [
                go.Scatter(
                    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': go.Layout(
                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)
```

Visualisation

- On obtient le graphique interactif suivant :

Visualisation



Markdown

- Bien que Dash implémente HTML via la librairie dash_html_components, il peut être laborieux de tout rédiger en HTML.

- Pour écrire des blocs de texte, il est plus pratique d'utiliser l'élément Markdown de la librairie dash_core_components.

Markdown

- Exemple d’utilisation :

PYTHON

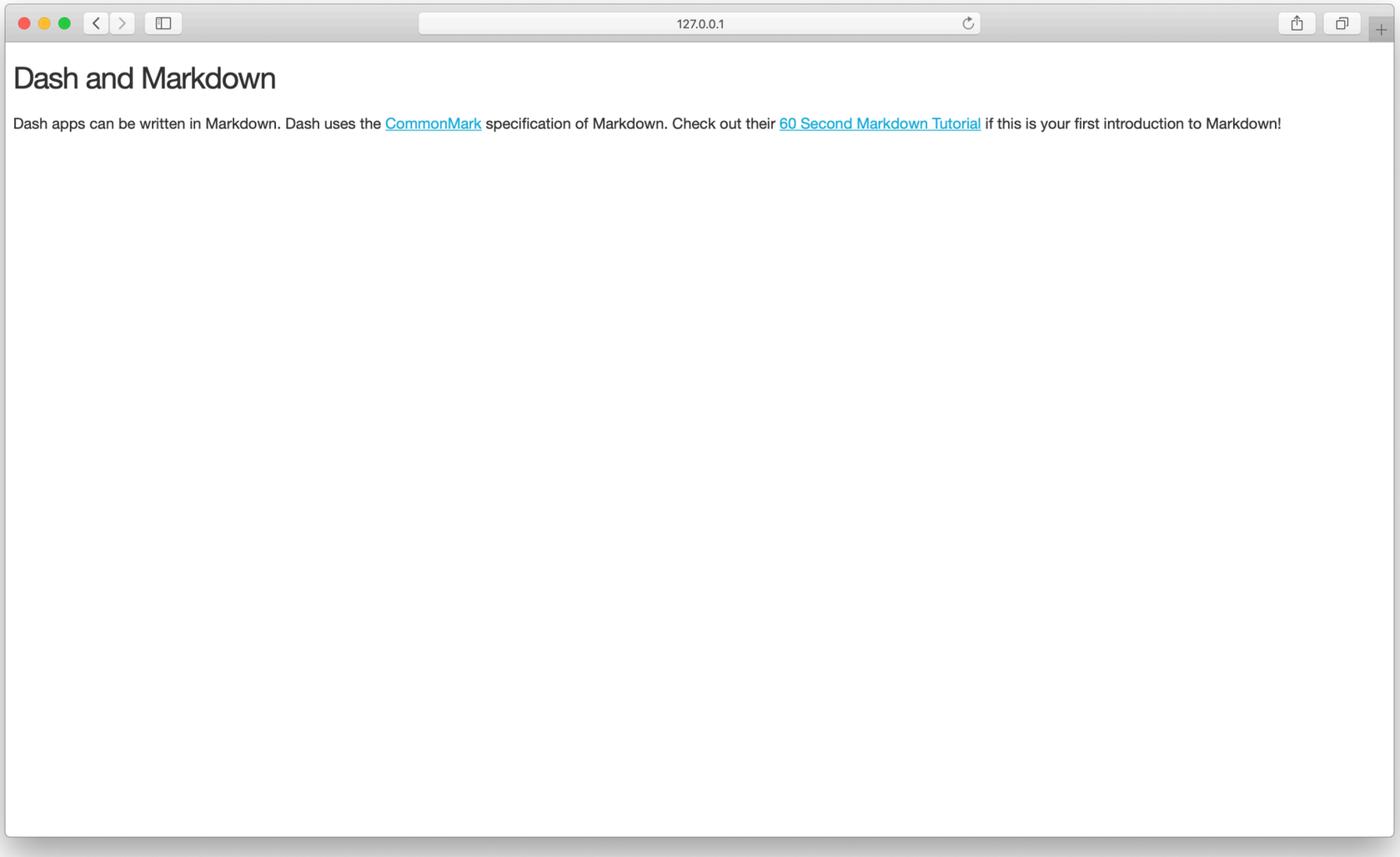
```
import dash
import dash_core_components as dcc
import dash_html_components as html

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

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

Markdown

```
markdown_text = '''
### Dash and Markdown

Dash apps can be written in Markdown.
Dash uses the [CommonMark](http://commonmark.org/)
specification of Markdown.
Check out their [60 Second Markdown Tutorial](http://commonmark.org/
help/)
if this is your first introduction to Markdown!
'''

app.layout = html.Div([
    dcc.Markdown(children=markdown_text)
])

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

Markdown

- A l’exécution, on obtient :



Core Components

- La libraire dash_core_components fournit des éléments de haut niveau, tels que des menus déroulants, des graphiques, des blocs markdown, etc.

- Comme toujours sous Dash, ils sont spécifiés déclarativement ; chaque option configurable est disponible via les arguments mots-clefs du élément.

- Pour accéder à l'ensemble des éléments disponibles, consulter : https://dash.plot.ly/dash-core-components.

- Voici un exemple d'utilisation :

Core Components

PYTHON

```
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)
```

Core Components

```
app.layout = 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
    ),
```

Core Components

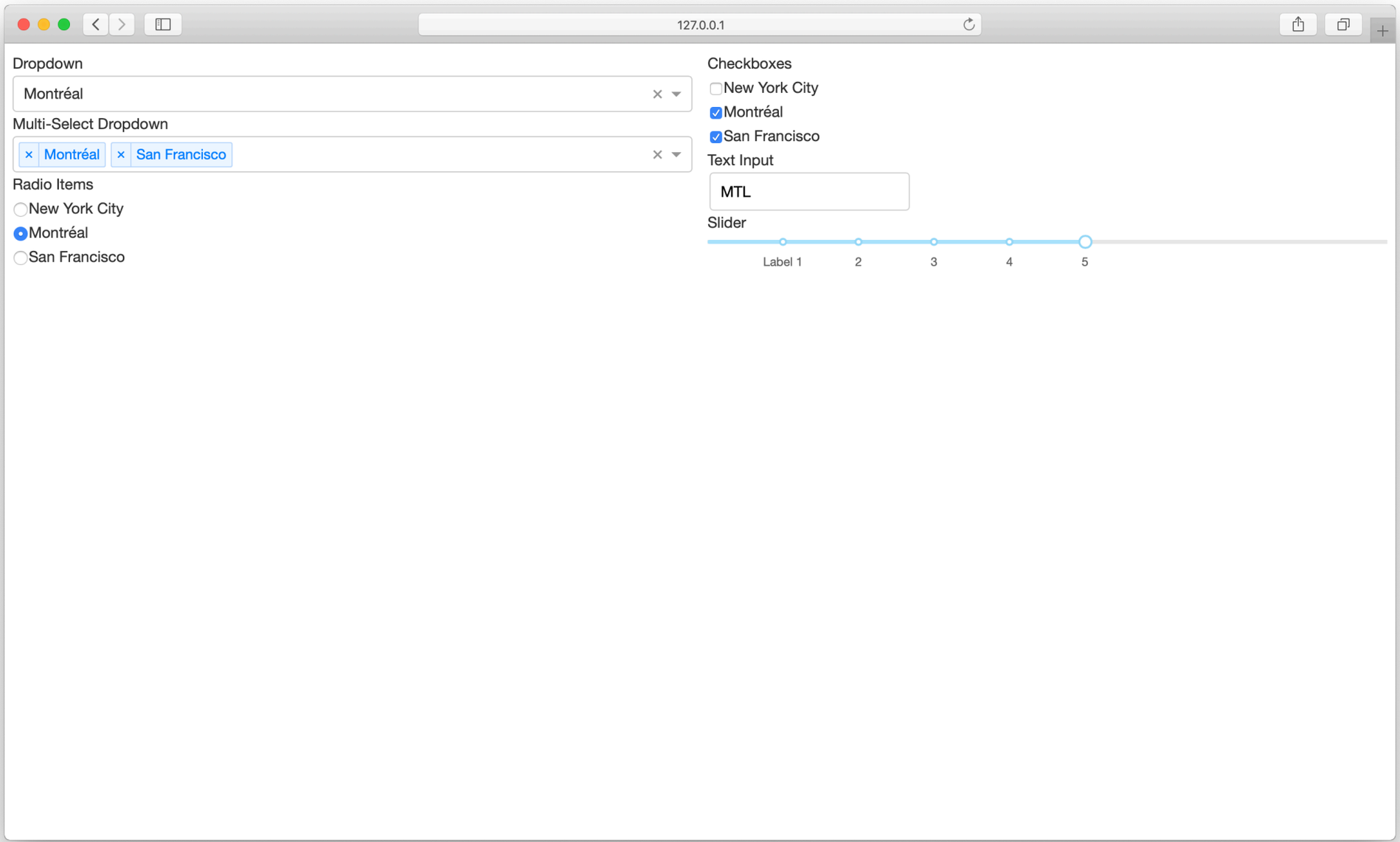
```
html.Label('Radio Items'),
 dcc.RadioItems(
     options=[
         {'label': 'New York City', 'value': 'NYC'},
         {'label': u'Montréal', 'value': 'MTL'},
         {'label': 'San Francisco', 'value': 'SF'}
     ],
     value='MTL'
),

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

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

Core Components

```
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})

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

- On obtient le résultat suivant :

Core Components



L’aide

- Les éléments de Dash sont utilisés déclarativement : chaque aspect configurable d’un élément peut être spécifié lors de l’instanciation à l’aide d’un argument mot-clef.

- Pour obtenir de l’aide sur un élément et ses arguments, on peut utiliser la fonction help.

- Exemple :

L’aide

```
help(dcc.Dropdown)
```

```
Help on class Dropdown in module dash_core_components.Dropdown:

class Dropdown(dash.development.base_component.Component)
 |  Dropdown(id=undefined, options=undefined, value=undefined, className=undefined, clearable=undefined,
disabled=undefined, multi=undefined, placeholder=undefined, searchable=undefined, style=undefined,
loading_state=undefined, **kwargs)
 |
 |  A Dropdown component.
 |  Dropdown is an interactive dropdown element for selecting one or more
 |  items.
 |  The values and labels of the dropdown items are specified in the `options`
 |  property and the selected item(s) are specified with the `value` property.

… etc.
```

Conclusion

- Le « layout » d'une application Dash contrôle son apparence.

- Ce « layout » est un arbre hiérarchique d'éléments.

- La librairie dash_html_components fournit des classes correspondant à toutes les balises HTML et les arguments mots-clefs tels que style, className et id correspondent aux attributs HTML.

Conclusion

- La librairie dash_core_components permet d'implémenter des contrôles et de générer des graphiques.

- Pour plus de détails, voir : https://dash.plot.ly/dash-core-components et https://dash.plot.ly/dash-html-components.

Tableaux de bord avec Python Dash

2. Fonctions de rappel

Salim Lardjane
Université de Bretagne Sud

Première application

- Dans la suite, nous allons voir comment rendre les applications Dash **interactives**.

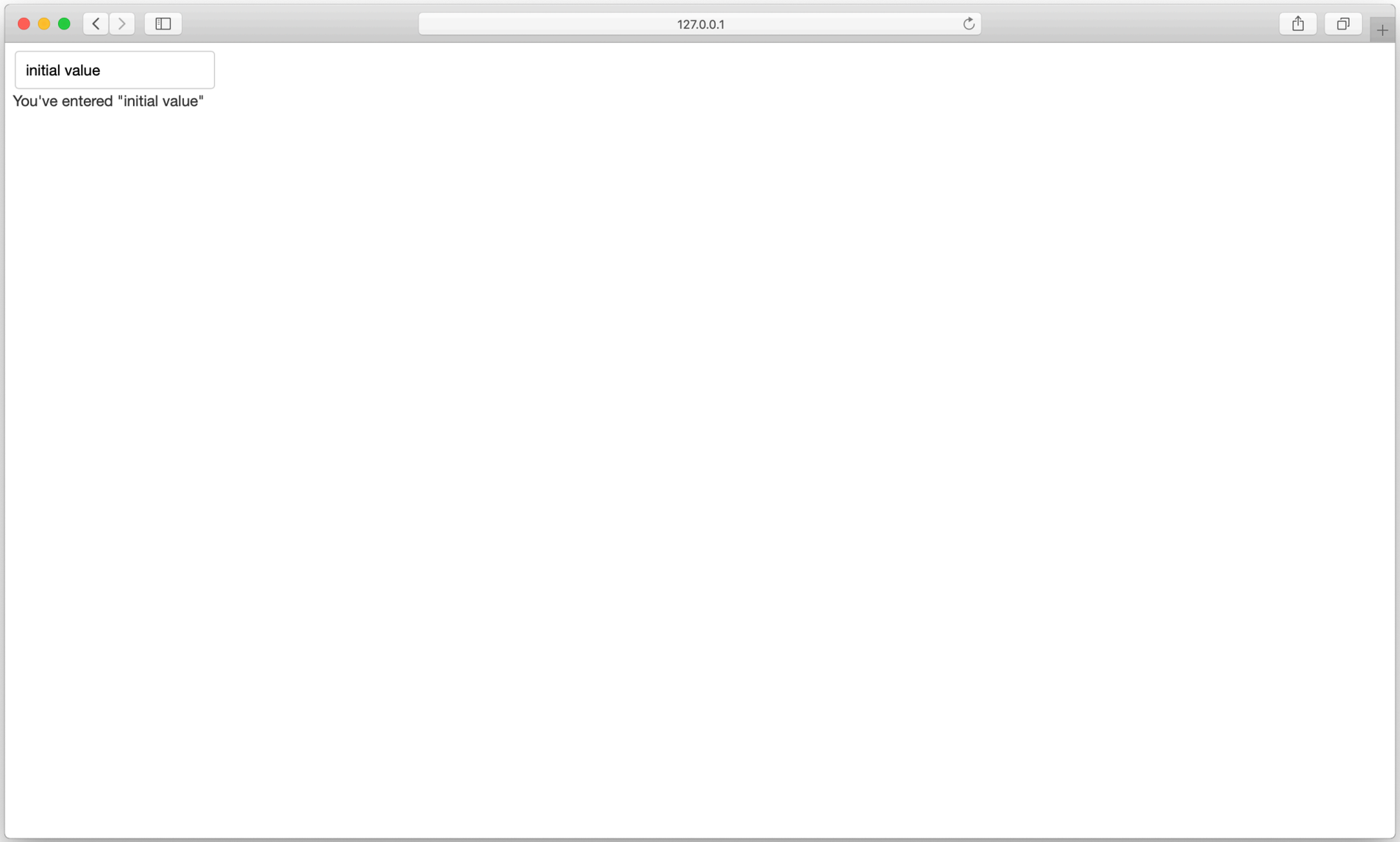
- Commençons par un exemple simple :

Première application

PYTHON

```
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)
```

Première application



Première application

- Si l’on saisit quelque chose dans le champ de saisie, le message en dessous est mis à jour en temps réel.

- Examinons précisément comment cela fonctionne.

Première application

- Les « entrées » et les « sorties » de notre applications sont décrites déclarativement à l'aide de la **fonction de rappel** app.callback.

- *Sous Dash, les « entrées » et les « sorties » de l'application sont toujours des propriétés d'un élément.*

- Dans l'exemple, l'entrée est la propriété « value » de élément d'identifiant « my-id ».

- La « sortie » est la propriété « children » de l'élément d'identifiant « my-div ».

Première application

- Chaque fois que la propriété associée à une « entrée » change, la fonction de rappel est appelée automatiquement.

- Dash fournit à la fonction la nouvelle valeur de la propriété en « entrée » et Dash met à jour la propriété du élément de « sortie » avec ce qui est renvoyé par la fonction.

- Les mots-clefs component-id et component-property sont optionnels. On les a utilisé ici pour améliorer la lisibilité du programme.

Première application

- Notons que nous ne donnons pas de valeur à la propriété chidren du élément my-div dans le layout.
- Lorsque l’application Dash démarre, elle appelle automatiquement tous les callbacks avec les valeurs initiales des éléments d’entrée afin de renseigner l’état initial des éléments de sortie.

Première application

- Si l'on avait spécifié quelque chose du type `html.Div(id='my-div', children='Hello world')`, cela aurait été écrasé au démarrage de l'application.
- C'est un peu comme sous Microsoft Excel : dès que le contenu d'une cellule est modifié, cela est répercuté sur toutes les cellules qui en dépendent. On appelle cela la « programmation réactive ».

Première application

- Chaque élément est décrit initialement à l'aide de ses arguments mots-clefs, qui spécifient ses propriétés.
- Ces propriétés peuvent être mises à jour dynamiquement via une **fonction de rappel** (ang. callback fonction).

Première application

- Souvent, on met à jour la propriété children d'un élément pour afficher un nouveau texte, ou la propriété figure d'un élément dcc.Graph pour afficher de nouvelles données, mais on peut également mettre à jour la propriété style d'un élément ou même les options disponibles d'un élément dcc.Dropdown (menu déroulant).

Deuxième exemple

- Considérons à présent un deuxième exemple, où une règle dcc.Slider est utilisée pour mettre à jour un graphique dcc.Graph :

Deuxième exemple

PYTHON

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

import pandas as pd
import plotly.graph_objs as go

df = pd.read_csv(
    'https://raw.githubusercontent.com/plotly/'
    'datasets/master/gapminderDataFiveYear.csv')

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

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

Deuxième exemple

```
app.layout = html.Div([
    dcc.Graph(id='graph-with-slider'),
    dcc.Slider(
        id='year-slider',
        min=df['year'].min(),
        max=df['year'].max(),
        value=df['year'].min(),
        marks={str(year): str(year) for year in df['year'].unique()},
        step=None
    )
])
```

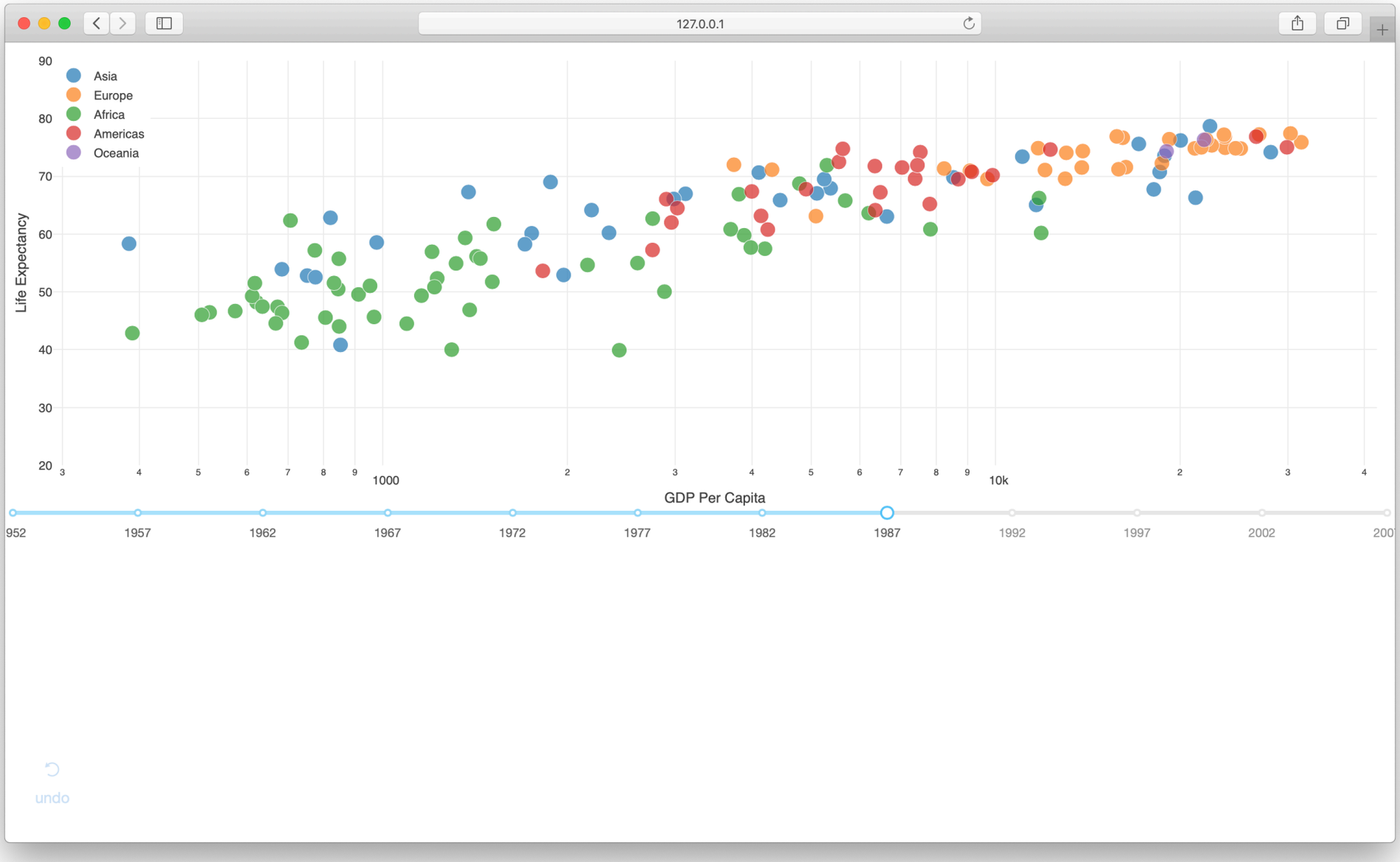
Deuxième exemple

```
@app.callback(
    Output('graph-with-slider', 'figure'),
    [Input('year-slider', 'value')])
def update_figure(selected_year):
    filtered_df = df[df.year == selected_year]
    traces = []
    for i in filtered_df.continent.unique():
        df_by_continent = filtered_df[filtered_df['continent'] == i]
        traces.append(go.Scatter(
            x=df_by_continent['gdpPercap'],
            y=df_by_continent['lifeExp'],
            text=df_by_continent['country'],
            mode='markers',
            opacity=0.7,
            marker={
                'size': 15,
                'line': {'width': 0.5, 'color': 'white'}
            },
            name=i
        ))
```

Deuxième exemple

```
return {
        'data': traces,
        'layout': go.Layout(
            xaxis={'type': 'log', 'title': 'GDP Per Capita'},
            yaxis={'title': 'Life Expectancy', 'range': [20, 90]},
            margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
            legend={'x': 0, 'y': 1},
            hovermode='closest'
        )
    }


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

Deuxième exemple



Deuxième exemple

- Dans cet exemple, la propriété « value » de l'élément Slider est l'entrée de l'application et sa sortie est la propriété « figure » de l'élément Graph.

- Dès que value est modifiée, Dash appelle la fonction de rappel update_figure en lui passant comme argument la nouvelle valeur de value.

- Cette fonction filtre le tableau de données à l'aide de cette nouvelle valeur, construit un objet figure et le renvoie à l'application Dash.

Deuxième exemple

- L’exemple permet de mettre en évidence quelques points importants :

- On utilise la librairie Pandas pour importer et filtrer les données dans la mémoire.

- On importe les données au début de l’application. Le data frame obtenu est une variable globale de l’application et peut être accédé à partir de la fonction de rappel.

- Charger les données en mémoire peut être coûteux. C’est pourquoi on le fait au début de l’application plutôt qu’à l’intérieur de la fonction de rappel.

Deuxième exemple

- La fonction de rappel ne modifie les données initiales ; elle se contente de créer des copies filtrées de celles-ci, ce qui est important.

- *Une règle générale est que les fonctions de rappel ne doivent pas modifier les variables se situant hors de leur portée.*

- Si ce n’est pas le cas, une session utilisateur peut influencer la suivante. De plus, si l’application est déployée en parallèle, les modifications sur un thread ne seront pas répercutées sur les autres.

Entrées multiples

- Sous Dash, toute sortie peut avoir des éléments en entrée multiples.

- Dans l'exemple suivant, on regroupe cinq entrées (la propriété value de deux menus déroulants, deux éléments de boutons radio et un élément règle) pour obtenir une sortie (la propriété figure de l'élément Graph).

- Bien noter que la fonction de rappel app.callback regroupe les cinq dash.dependencies.Input au sein d'une liste passée en deuxième argument.

Entrées multiples

PYTHON

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

import pandas as pd
import plotly.graph_objs as go

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/'
    'cb5392c35661370d95f300086accea51/raw/'
    '8e0768211f6b747c0db42a9ce9a0937dafcbd8b2/'
    'indicators.csv')
```

Entrées multiples

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

app.layout = html.Div([
    html.Div([

        html.Div([
            dcc.Dropdown(
                id='xaxis-column',
                options=[{'label': i, 'value': i} for i in available_indicators],
                value='Fertility rate, total (births per woman)'
            ),
            dcc.RadioItems(
                id='xaxis-type',
                options=[{'label': i, 'value': i} for i in ['Linear', 'Log']],
                value='Linear',
                labelStyle={'display': 'inline-block'}
            )
        ],
        style={'width': '48%', 'display': 'inline-block'}),
```

Entrées multiples

```
html.Div([
        dcc.Dropdown(
            id='yaxis-column',
            options=[{'label': i, 'value': i} for i in available_indicators],
            value='Life expectancy at birth, total (years)'
        ),
        dcc.RadioItems(
            id='yaxis-type',
            options=[{'label': i, 'value': i} for i in ['Linear', 'Log']],
            value='Linear',
            labelStyle={'display': 'inline-block'}
        )
    ],style={'width': '48%', 'float': 'right', 'display': 'inline-block'})
]),

dcc.Graph(id='indicator-graphic'),
```

Entrées multiples

```
dcc.Slider(
    id='year--slider',
    min=df['Year'].min(),
    max=df['Year'].max(),
    value=df['Year'].max(),
    marks={str(year): str(year) for year in df['Year'].unique()},
    step=None
)
])
```

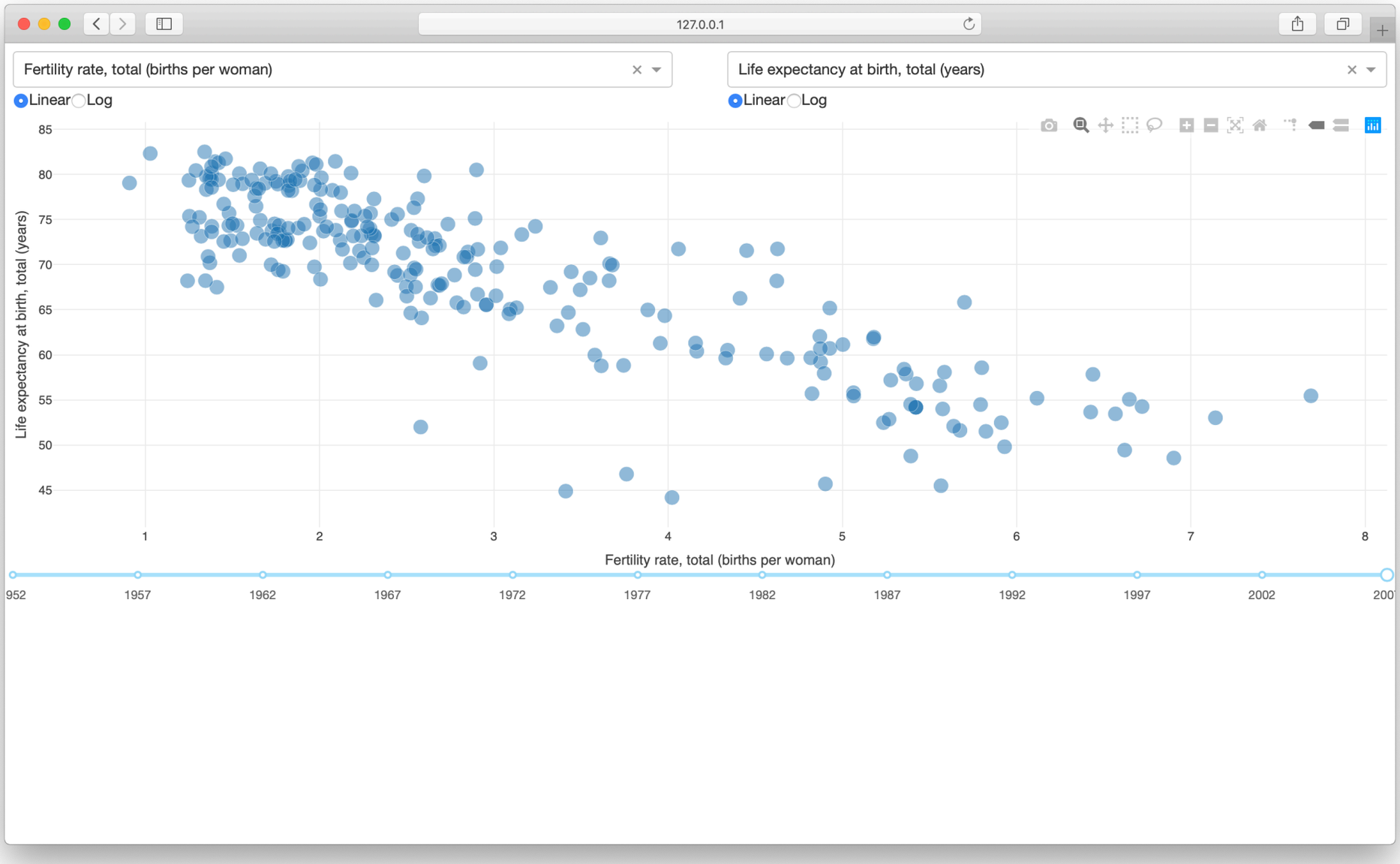
Entrées multiples

```
@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': [go.Scatter(
        x=dff[dff['Indicator Name'] == xaxis_column_name]['Value'],
        y=dff[dff['Indicator Name'] == yaxis_column_name]['Value'],
        text=dff[dff['Indicator Name'] == yaxis_column_name]['Country Name'],
        mode='markers',
        marker={
            'size': 15,
            'opacity': 0.5,
            'line': {'width': 0.5, 'color': 'white'}
        }
    )],
```

Entrées multiples

```
'layout': go.Layout(
    xaxis={
        'title': xaxis_column_name,
        'type': 'linear' if xaxis_type == 'Linear' else 'log'
    },
    yaxis={
        'title': yaxis_column_name,
        'type': 'linear' if yaxis_type == 'Linear' else 'log'
    },
    margin={'l': 40, 'b': 40, 't': 10, 'r': 0},
    hovermode='closest'
)
}


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

Entrées multiples



Entrées multiples

- Dans l'exemple précédent, la fonction update_graph est appelée à chaque fois que la propriété value de l'un au moins des éléments Dropdown, Slider ou RadioItems est modifiée.

- Généralisons à présent au cas des sorties multiples.

Sorties multiples

- Jusqu'ici, toutes les fonctions de rappel qu'on a programmé ne mettaient à jour qu'une seule propriété en sortie.

- On peut, en fait, en mettre à jour plusieurs à la fois : il suffit pour cela de lister les propriétés souhaitées dans le décorateur et renvoyer autant d'éléments lors du rappel.

- C'est particulièrement pratique si deux sorties dépendent d'un résultat intermédiaire coûteux, par exemple d'une requête de données de temps d'exécution important.

- Exemple :

Sorties multiples

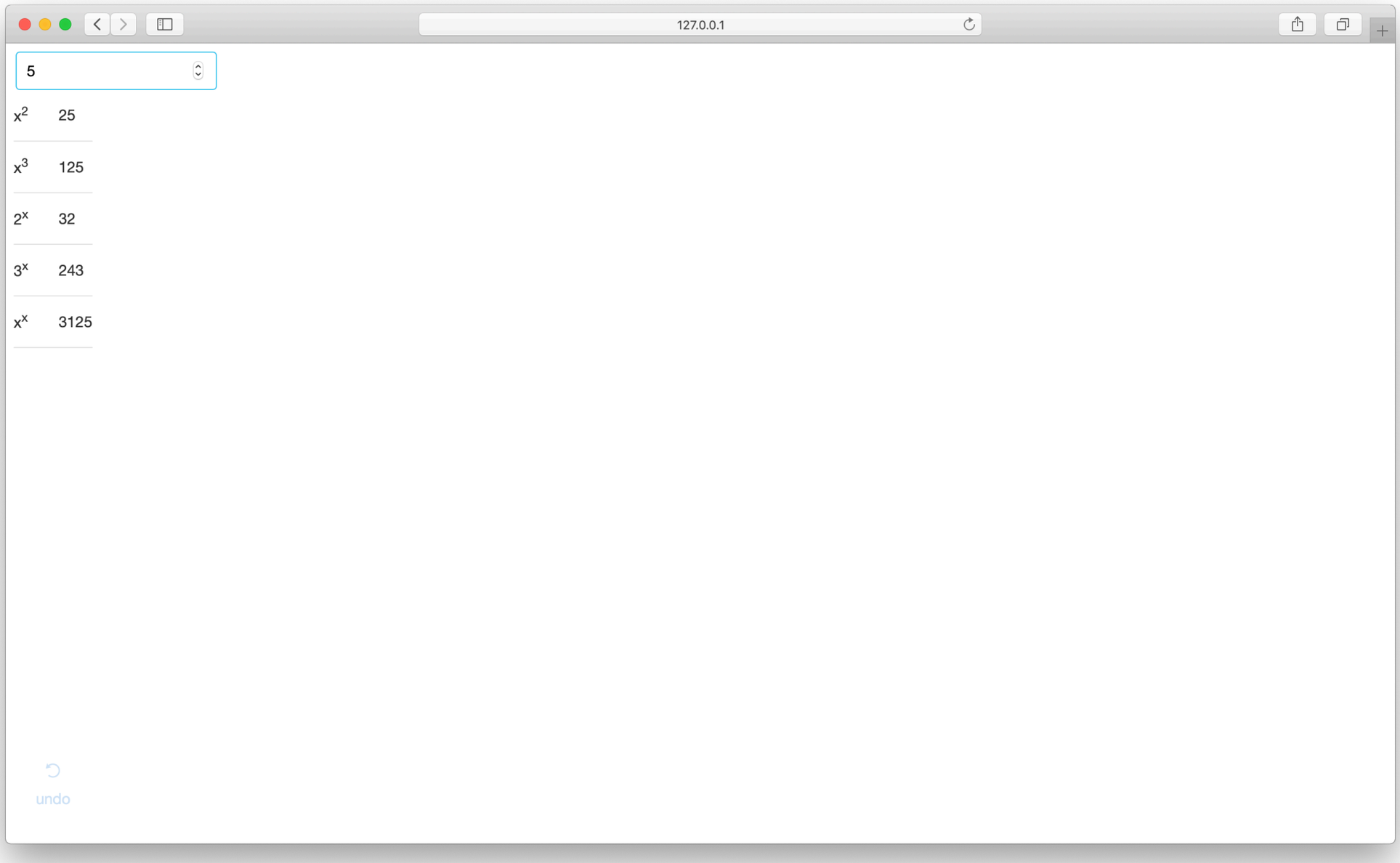
PYTHON

```
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='num',
        type='number',
        value=5
    ),
    html.Table([
        html.Tr([html.Td(['x', html.Sup(2)]), html.Td(id='square')]),
        html.Tr([html.Td(['x', html.Sup(3)]), html.Td(id='cube')]),
        html.Tr([html.Td([2, html.Sup('x')]), html.Td(id='twos')]),
        html.Tr([html.Td([3, html.Sup('x')]), html.Td(id='threes')]),
        html.Tr([html.Td(['x', html.Sup('x')]), html.Td(id='x^x')]),
    ]),
])
```

Sorties multiples

```
@app.callback(
    [Output('square', 'children'),
     Output('cube', 'children'),
     Output('twos', 'children'),
     Output('threes', 'children'),
     Output('x^x', 'children')],
    [Input('num', 'value')])
def callback_a(x):
    return x**2, x**3, 2**x, 3**x, x**x


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

Sorties multiples



Sorties multiples

- Il y a quelques précautions à prendre avant de regrouper des sorties :
- Si les sorties n’ont que quelques entrées en commun, les maintenir séparés peut éviter des mises à jour inutiles.
- Si les sorties ont exactement les mêmes entrées mais sont obtenues à l’aide de calculs différents, utiliser des fonctions de rappel différentes peut permettre de faire les calculs en parallèle.

Rappels enchaînés

- On peut enchaîner sorties et entrées : la sortie d'une fonction de rappel peut être l'entrée d'une autre fonction de rappel.

- Ce mécanisme peut être utilisé pour créer des IU dynamiques, où un élément d'entrée met à jour les options disponibles dans l'élément d'entrée suivant.

- Exemple :

Rappels enchaînés

PYTHON

```
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)

all_options = {
    'America': ['New York City', 'San Francisco', 'Cincinnati'],
    'Canada': [u'Montréal', 'Toronto', 'Ottawa']
}
```

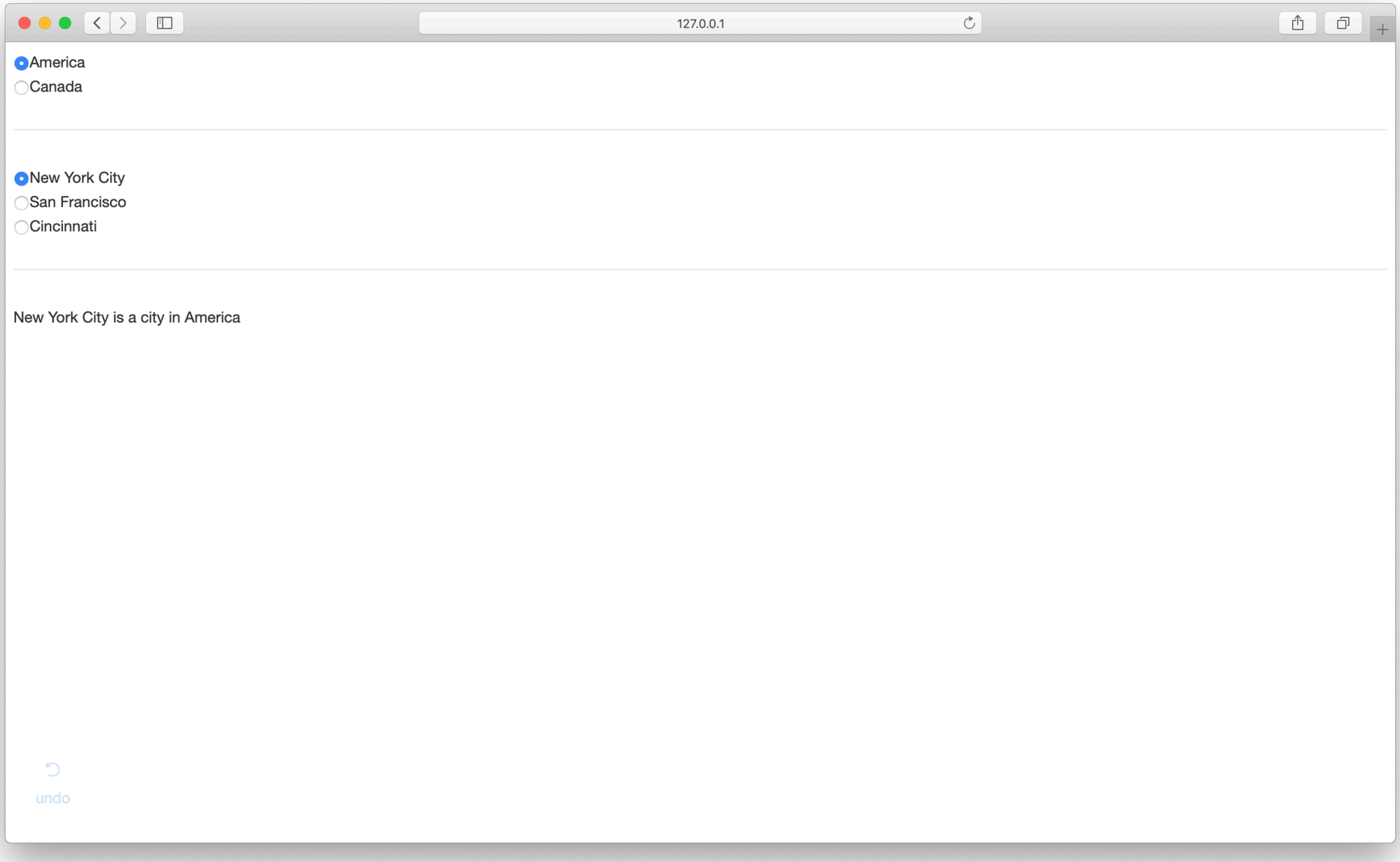
Rappels enchaînés

```
app.layout = html.Div([
    dcc.RadioItems(
        id='countries-dropdown',
        options=[{'label': k, 'value': k} for k in all_options.keys()],
        value='America'
    ),

    html.Hr(),

    dcc.RadioItems(id='cities-dropdown'),

    html.Hr(),

    html.Div(id='display-selected-values')
])
```

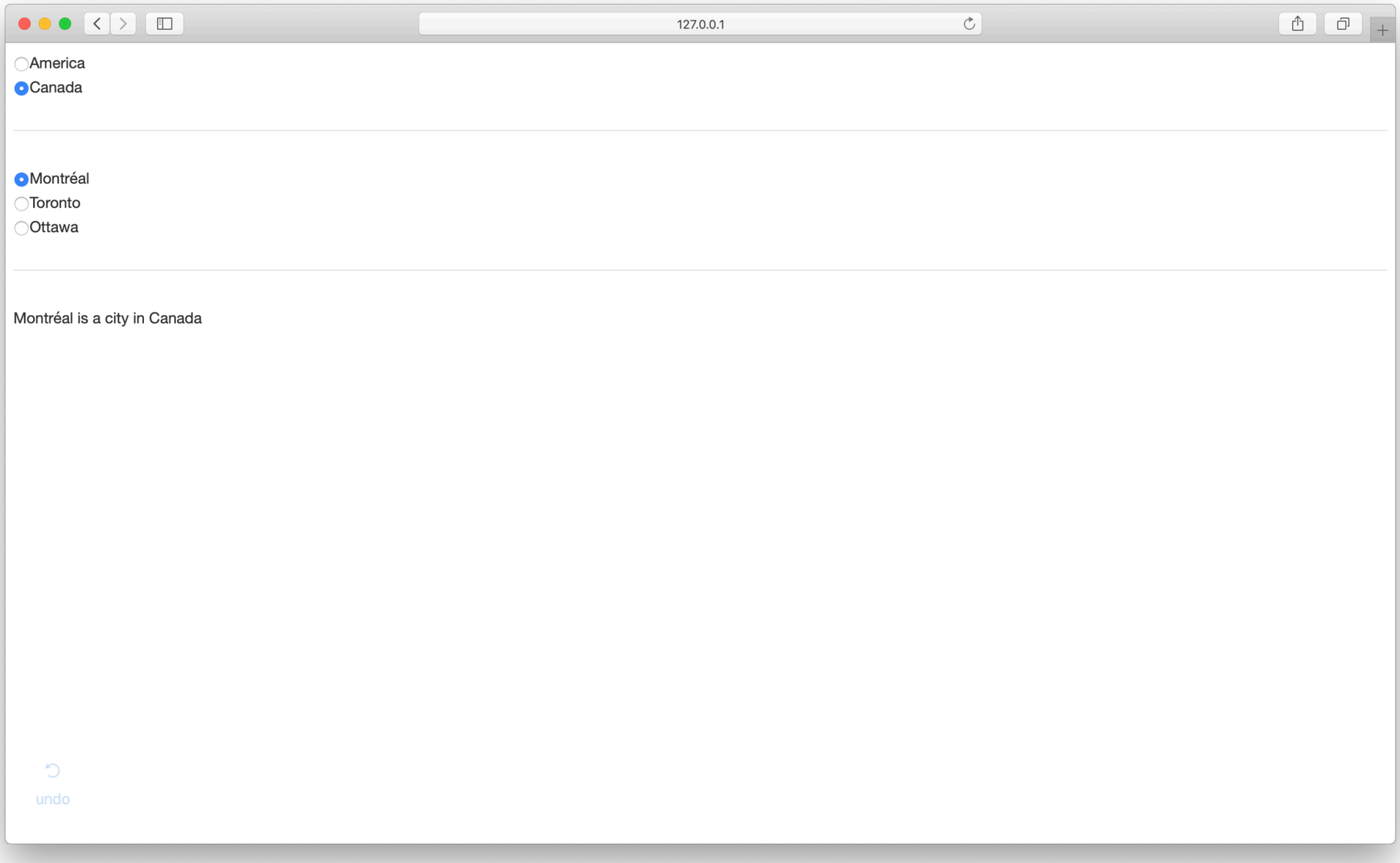
Rappels enchaînés

```
@app.callback(
    Output('cities-dropdown', 'options'),
    [Input('countries-dropdown', 'value')])
def set_cities_options(selected_country):
    return [{'label': i, 'value': i} for i in all_options[selected_country]]


@app.callback(
    Output('cities-dropdown', 'value'),
    [Input('cities-dropdown', 'options')])
def set_cities_value(available_options):
    return available_options[0]['‘value']

@app.callback(
    Output('display-selected-values', 'children'),
    [Input('countries-dropdown', 'value'),
     Input('cities-dropdown', 'value')])
def set_display_children(selected_country, selected_city):
    return u'{} is a city in {}'.format(
        selected_city, selected_country,
    )


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

Rappels enchaînés



Rappels enchaînés



Rappels enchaînés

- Le premier rappel met à jour les options disponibles dans le deuxième élément RadioItems en se basant sur la valeur sélectionnée dans le premier élément RadioItems.

- Le deuxième rappel fixe une valeur initiale lorsque la propriété options est modifiée : cette valeur initiale est la première valeur dans le tableau des options.

- Le dernier rappel affiche la valeur de chaque élément. Si on modifie la propriété value de l'élément RadioItems pays, Dash attend que la propriété value de l'élément villes soit mise à jour avant d'appeler la dernière fonction de rappel. Cela évite d'avoir des rappels incohérents, comme « USA » avec « Montréal ».

Références

- Pour plus de détails et des compléments, on pourra consulter le « Dash User Guide » : https://dash.plot.ly.