Dash Callbacks

- Los Callbacks son funciones de python que se llaman automáticamente cuando algún componente cambia.
- Nos permiten generar interactividad en nuesta aplicación.

```
app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),

])
```

```
@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)
```

- Las entradas y las salidas del callback son descritas de forma declarativa en los argumentos del decorador @app.callback.
- Estas entradas y salidas son propiedades de un componente en particular.
- La entrada es el valor del componete con id my-input.
- La salida es el children (es decir el contenido) del componente con id my-output.
- Cuando la propiedad de la entrada cambia, la función es llamada automáticamente.
- Dash introduce el nuevo valor en el argumento de la función.
- Las keywords component_id y component_property so opcionales y normalmente no se ponen.

- Output y Input se tienen que importar:

```
from dash.dependencies import Input, Output
```

- No se define un valor para los componentes de salida, por que cuando DASH es iniciado llama a todos los callbacks de manera automática con los valores iniciales de los componentes de entrada.
- Este paradigma de programación se conoce como "Reactive Programming".
- Todos los componttes en dash se definen con una lista de argumentos keyword, todas esas propiedades pueden ser cambiadas dinámicamente.

```
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
    )
])
```

```
@app.callback(
    Output('graph-with-slider', 'figure'),
    Input('year-slider', 'value'))
def update_figure(selected_year):
    filtered_df = df[df.year == selected_year]
    fig = px.scatter(filtered_df, x="gdpPercap", y="lifeExp",
                     size="pop", color="continent", hover_name="country",
                     log_x=True, size_max=55)
    fig.update_layout(transition_duration=500)
    return fig
```

- En este ejemplo la propiedad value del Slider es la entrada.
- La salida es la propiedad figure de Graph.
- Cuando el valor del slider cambia la figura se actualiza filtrando el dataframe con el valor del slider.

Algunas consideraciones:

- Estamos usando pandas para filtrar y cargar los datos en memoria.
- Los datos los tenemos en una variable global para que las funciones callback puedan acceder a ellos.
- Cargar datos en memoria es costoso; lo realizamos al principio para que no tener que hacerlo más veces.
- El callback no modifica los datos originales.
- Los callbacks nunca tienen que modificar variables fuera de su scope. Hacerlo puede ocasionar que usuarios distintos entren en conflicto.
- Las transiciones: layout.transition hacen animación al pasar de una figura a otra.

- En Dash una "Output" puede tener múltiples "Input".
- Veamos `example_3.py`:

```
@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):
    ...
```

- Los valores de 2 Dropdown, 2 RadioItems, y 1 Slider modifican un componente de salida, en este caso una figura.
- La función update_graph es llamada cuando uno de los componentes de entrada cambia.
- Los argumentos de la función update_graph se declaran en el mismo orden que en app.callback.

- También se pueden definir dentro de una lista:

```
@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):
    ...
```

Callbacks con múltiples salidas

- Un Callback puede modificar más de una propiedad de salida.
- Un ejemplo en `example_4.py`.

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

- Nuestro callback tendrá que retornar tantos valores como salidas.

- Esto es interesante cuando dos salidas dependen de un cálculo o una petición.
- En algunos casos no es buena idea:
  - Si las salidas dependen solo en algunas de las entradas.
  - Si el cálculo es distinto, tenerlo separado puede ser una ventaja ya que los callbacks se ejecutan en paralelo.

Callbacks encadenados

- Dash permite que la salida de un callback sea la entrada de otro.
- Esto permite crear cadenas dinámicas, donde la entrada de un componente actualiza las opciones del siguiente.
- Puedes ver un ejemplo en `example_5.py`.

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


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

- El primer callback actualiza las opciones del RadioItems countries-radio basandose en la entrada.
- El segundo callback selecciona la primera opción disponible para el cities-radio RadioItems.
- El callback final muestra las opciones seleccionadas para cada componente.
- Dash espera a que los valores sean actualizados de todos los callbacks antes de llamar al siguiente callback. Esto previene resultados inconsistentes.

Aplicaciones con estado

- En algunos casos es necesario leer valores de algún componente.
- Por ejemplo en un formulario.
- Normalmente solo queremos esto cuando el usuario ha terminado de introducir los datos.
- Si asisgnamos un callback directamente el comportamiento puede no ser el deseado.
- Podemos verlo en `example_6.py`

- dash.dependencies.State permite pasar valores sin ejecuciones de callbacks.

```
@app.callback(Output('output-state', 'children'),
              Input('submit-button-state', 'n_clicks'),
              State('input-1-state', 'value'),
              State('input-2-state', 'value'))
def update_output(n_clicks, input1, input2):
    ...
```

- En este ejemplo cambiar los valores en las cajas no ejecuta el callback.
- El callback solo es ejecutado al pulsar el botón.
- Al usar el botón podemos acceder a ellos.
- El callback escucha la propiedad n_clicks del html.Button, que es incrementada cada vez que se pulsa.