plotly

Star 23,446

Dash Python > ***Pattern-Matching Callbacks***

Plotly Studio: Transform any dataset into an interactive data application in minutes with AI. **Sign up for early access now.**

Pattern-Matching Callbacks

To get the most out of this page, make sure you've read about **Basic Callbacks** in the Dash Fundamentals.

The pattern-matching callback selectors `MATCH`, `ALL`, & `ALLSMALLER` allow you to write callbacks that respond to or update an arbitrary or dynamic number of components.

Simple Example with `ALL`

This example uses **partial property updates**, introduced in Dash 2.9. For an example that works with earlier versions of Dash, see Simple `ALL` Example Without Partial Updates at the end of this example.

This example renders an arbitrary number of `dcc.Dropdown` elements and the callback is fired whenever any of the `dcc.Dropdown` elements change. Try adding a few dropdowns and selecting their values to see how the app updates.

```
from dash import Dash, dcc, html, Input, Output, ALL, Patch, callback

app = Dash()

app.layout = html.Div(
    [
        html.Button("Add Filter", id="add-filter-btn", n_clicks=0),
        html.Div(id="dropdown-container-div", children=[]),
        html.Div(id="dropdown-container-output-div"),
    ]
)


@callback(
    Output("dropdown-container-div", "children"), Input("add-filter-btn", "n_clicks")
)
def display_dropdowns(n_clicks):
    patched_children = Patch()
    new_dropdown = dcc.Dropdown(
        ["NYC", "MTL", "LA", "TOKYO"],
        id={"type": "city-filter-dropdown", "index": n_clicks},
    )
    patched_children.append(new_dropdown)
    return patched_children


@callback(
    Output("dropdown-container-output-div", "children"),
    Input({"type": "city-filter-dropdown", "index": ALL}, "value"),
)
def display_output(values):
    return html.Div(
        [html.Div(f"Dropdown {i + 1} = {value}") for (i, value) in enumerate(values)]
    )
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

ADD FILTER

Select...

Dropdown 1 = None

Some notes about this example:

- Notice how the `id` in `dcc.Dropdown` is a *dictionary* rather than a *string*. This is a new feature that we enabled for pattern-matching callbacks (previously, IDs had to be strings).
- In our second callback, we have `Input({'type': 'city-filter-dropdown', 'index': ALL}, 'value')`. This means "match any input that has an ID dictionary where `'type'` is `'city-filter-dropdown'` and `'index'` is *anything*. Whenever the `value` property of any of the dropdowns change, send *all* of their values to the callback."
- The keys & values of the ID dictionary (`type`, `index`, `city-filter-dropdown`) are arbitrary. This could've be named `{'foo': 'bar', 'baz': n_clicks}`.
- However, for readability, we recommend using keys like `type`, `index`, or `id`. `type` can be used to refer to the class or set dynamic components and `index` or `id` could be used to refer *which* component you are matching within that set. In this example, we just have a single set of dynamic components but you may have multiple sets of dynamic components in more complex apps or if you are using `MATCH` (see below).
- In fact, in this example, we didn't actually *need* `'type': 'city-filter-dropdown'`. The same callback would have worked with `Input({'index': ALL}, 'value')`. We included `'type': 'city-filter-dropdown'` as an extra specifier in case you create multiple sets of dynamic components.
- The component properties themselves (e.g. `value`) cannot be matched by a pattern, only the IDs are dynamic.
- This example uses `Patch` to make a **partial update** to the `'children'` property of `'dropdown-container-div'`. We append a dropdown each time the first callback runs.

▶ Simple `ALL` Example Without Partial Updates

Simple Example with `MATCH`

> This example uses **partial property updates**, introduced in Dash 2.9. For an example that works with earlier versions of Dash, see Simple `MATCH` Example Without Partial Updates at the end of this example.

Like `ALL`, `MATCH` will fire the callback when any of the component's properties change. However, instead of passing *all* of the values into the callback, `MATCH` will pass just a single value into the callback. Instead of updating a single output, it will update the dynamic output that is "matched" with.

```
from dash import Dash, dcc, html, Input, Output, State, MATCH, Patch, callback

app = Dash()

app.layout = html.Div([
    html.Button("Add Filter", id="dynamic-add-filter-btn", n_clicks=0),
    html.Div(id='dynamic-dropdown-container-div', children=[]),
])

@callback(
    Output('dynamic-dropdown-container-div', 'children'),
    Input('dynamic-add-filter-btn', 'n_clicks')
    )
def display_dropdowns(n_clicks):
    patched_children = Patch()

    new_element = html.Div([
        dcc.Dropdown(
            ['NYC', 'MTL', 'LA', 'TOKYO'],
            id={
                'type': 'city-dynamic-dropdown',
                'index': n_clicks
```

```
            }
        ),
        html.Div(
            id={
                'type': 'city-dynamic-output',
                'index': n_clicks
            }
        )
    ])
    patched_children.append(new_element)
    return patched_children


@callback(
    Output({'type': 'city-dynamic-output', 'index': MATCH}, 'children'),
    Input({'type': 'city-dynamic-dropdown', 'index': MATCH}, 'value'),
    State({'type': 'city-dynamic-dropdown', 'index': MATCH}, 'id'),
)
def display_output(value, id):
    return html.Div(f"Dropdown {id['index']} = {value}")


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

ADD FILTER

Select...

Dropdown 0 = None

Notes about this example:

- The `display_dropdowns` callback returns two elements with the *same* `index`: a dropdown and a div.
- The second callback uses the `MATCH` selector. With this selector, we're asking Dash to:
    1. Fire the callback whenever the `value` property of any component with the id `'type': 'dynamic-dropdown'` changes: `Input({'type': 'dynamic-dropdown', 'index': MATCH}, 'value')`
    2. Update the component with the id `'type': 'dynamic-output'` and the `index` that *matches* the same `index` of the input: `Output({'type': 'dynamic-output', 'index': MATCH}, 'children')`
    3. Pass along the `id` of the dropdown into the callback: `State({'type': 'dynamic-dropdown', 'index': MATCH}, 'id')`
- With the `MATCH` selector, only a *single* value is passed into the callback for each `Input` or `State`. This is unlike the previous example with the `ALL` selector where Dash passed *all* of the values into the callback.
- Notice how it's important to design IDs dictionaries that "line up" the inputs with outputs. The `MATCH` contract is that Dash will update whichever output has the same dynamic ID as the id. In this case, the "dynamic ID" is the value of the `index` and we've designed our layout to return dropdowns & divs with identical values of `index`.
- In some cases, it may be important to know *which* dynamic component changed. As above, you can access this by setting `id` as `State` in the callback.
- You can also use `dash.callback_context` to access the inputs and state and to know which input changed. `outputs_list` is particularly useful with `MATCH` because it can tell you *which* dynamic component this particular invocation of the callback is responsible for updating. Here is what that data might look like with two dropdowns rendered on the page after we change the first dropdown.
    - `dash.callback_context.triggered_prop_ids` (available from Dash 2.4) returns a dictionary of inputs that triggered the callback. Each key is a `<component_id>.<component_property>` and the corresponding value is the `<component_id>`. In this example, we can see that the id of the component that triggered the callback was `{'index': 0, 'type': 'dynamic-dropdown'}` and the property was `value`:

```
{
  '{"index":0,"type":"dynamic-dropdown"}.value': {
```

```
        "index": 0,
        "type": "dynamic-dropdown",
    }
}
```

- `dash.callback_context.triggered`. Note that the `prop_id` is a stringified dictionary with no whitespace.

```
[
  {
    'prop_id': '{"index":0,"type":"dynamic-dropdown"}.value',
    'value': 'NYC'
  }
]
```

- `dash.callback_context.inputs`. Note that the key is a stringified dictionary with no whitespace.

```
{
  '{"index":0,"type":"dynamic-dropdown"}.value': 'NYC'
}
```

- `dash.callback_context.inputs_list`. Each element of the list corresponds to one of the input declarations. If one of the input declarations matches a pattern then it will contain a list of values.

```
[
  [
    {
      'id': {
        'index': 0,
        'type': 'dynamic-dropdown'
      },
      'property': 'value',
      'value': 'NYC'
    }
  ]
]
```

- `dash.callback_context.outputs_list`

```
{
  'id': {
      'index': 0,
      'type': dynamic-output'
  },
  'property': 'children'
}
```

▶ Simple `MATCH` Example Without Partial Updates

Simple Example with `ALLSMALLER`

> This example uses **partial property updates**, introduced in Dash 2.9. For an example that works with earlier versions of Dash, see Simple `ALLSMALLER` Example Without Partial Updates at the end of this example.

In the following example, `ALLSMALLER` is used to pass in the values of all of the dropdowns on the page that have an index smaller than the index corresponding to the div.

The user interface in the example below displays filter results that are increasingly specific in each as we apply each additional dropdown.

`ALLSMALLER` can only be used in `Input` and `State` items, and must be used on a key that has `MATCH` in the `Output` item(s).

`ALLSMALLER` it isn't always necessary (you can usually use `ALL` and filter out the indices in your callback) but it will make your logic simpler.

```
from dash import Dash, dcc, html, Input, Output, Patch, MATCH, ALLSMALLER, callback
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

app = Dash(__name__, suppress_callback_exceptions=True)

app.layout = html.Div([
    html.Button('Add Filter', id='add-filter-ex3-btn', n_clicks=0),
    html.Div(id='container-ex3-div', children=[]),
])

@callback(
    Output('container-ex3-div', 'children'),
    Input('add-filter-ex3-btn', 'n_clicks'))
def display_dropdowns(n_clicks):
    patched_children = Patch()
    patched_children.append(html.Div([
        dcc.Dropdown(
            df['country'].unique(),
            df['country'].unique()[n_clicks],
            id={
                'type': 'filter-dd-ex3',
                'index': n_clicks
            }
        ),
        html.Div(id={
            'type': 'output-div-ex3',
            'index': n_clicks
        })
    ]))
    return patched_children


@callback(
    Output({'type': 'output-div-ex3', 'index': MATCH}, 'children'),
    Input({'type': 'filter-dd-ex3', 'index': MATCH}, 'value'),
    Input({'type': 'filter-dd-ex3', 'index': ALLSMALLER}, 'value'),
)
def display_output(matching_value, previous_values):
    previous_values_in_reversed_order = previous_values[::-1]
    all_values = [matching_value] + previous_values_in_reversed_order

    dff = df[df['country'].str.contains('|'.join(all_values))]
    avgLifeExp = dff['lifeExp'].mean()

    # Return a slightly different string depending on number of values
    if len(all_values) == 1:
        return html.Div('{:.2f} is the life expectancy of {}'.format(
            avgLifeExp, matching_value
        ))
    elif len(all_values) == 2:
```

ADD FILTER

Afghanistan

43.83 is the life expectancy of Afghanistan

- In the example above, try adding a few filters and then change the first dropdown. Notice how changing this dropdown will update the text of each `html.Div` that has an index that depends on that dropdown.
- That is, each `html.Div` will get updated whenever any of the dropdowns with an `index` smaller than it has changed.
- So, if there are 10 filters added and the first dropdown has changed, Dash will fire your callback 10 times, once to update each `html.Div` that depends on the `dcc.Dropdown` that changed.
- As above, you can also use `dash.callback_context` to access the inputs and state and to know which input changed. Here is what that data might look like when updating the second div with two dropdowns rendered on the page after we change the first dropdown.

- `dash.callback_context.triggered_prop_ids` (available from Dash 2.4) returns a dictionary of inputs that triggered the callback. Each key is a `<component_id>.<component_property>` and the corresponding value is the `<component_id>`. In this example, we can see that the id of the component that triggered the callback was `{'index': 0, 'type': 'filter-dropdown-ex3'}` and the property was `value`:

```
{
  '{"index":0,"type":"filter-dropdown-ex3"}.value': {
      "index": 0,
      "type": "filter-dropdown-ex3",
  }
}
```

- `dash.callback_context.triggered`. Note that the `prop_id` is a stringified dictionary with no whitespace.

```
[
  {
    'prop_id': '{"index":0,"type":"filter-dropdown-ex3"}.value',
    'value': 'Canada'
  }
]
```

- `dash.callback_context.inputs`. Note that the key is a stringified dictionary with no whitespace.

```
{
'{"index":1,"type":"filter-dropdown-ex3"}.value': 'Albania',
'{"index":0,"type":"filter-dropdown-ex3"}.value': 'Canada'
}
```

- `dash.callback_context.inputs_list`. Each element of the list corresponds to one of the input declarations. If one of the input declarations matches a pattern then it will contain a list of values.

```
[
  {
    'id': {
      'index': 1,
      'type': 'filter-dropdown-ex3'
    },
    'property': 'value',
    'value': 'Albania'
  },
  [
    {
      'id': {
        'index': 0,
        'type': 'filter-dropdown-ex3'
      },
      'property': 'value',
      'value': 'Canada'
    }
  ]
]
```

- `dash.callback_context.outputs_list`

```
{
  'id': {
      'index': 1,
      'type': output-ex3'
  },
  'property': 'children'
}
```

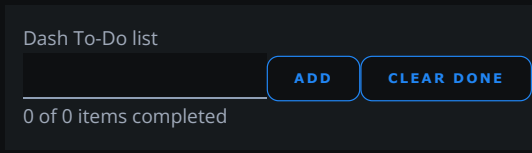
▶ `ALLSMALLER` Example Without Partial Updates

Todo App

Creating a Todo App is a classic UI exercise in that demonstrates many features in common "create, read, update and delete" (CRUD) applications.

> This example uses partial property updates and duplicate callback outputs, introduced in Dash 2.9. For an example that works with earlier versions of Dash, see Todo App Without Partial Updates below.

```
from dash import Dash, dcc, html, Input, Output, State, MATCH, ALL, Patch, callback

app = Dash()

app.layout = html.Div(
    [
        html.Div("Dash To-Do list"),
        dcc.Input(id="new-item-input"),
        html.Button("Add", id="add-btn"),
        html.Button("Clear Done", id="clear-done-btn"),
        html.Div(id="list-container-div"),
        html.Div(id="totals-div"),
    ]
)

# Callback to add new item to list
@callback(
    Output("list-container-div", "children", allow_duplicate=True),
    Output("new-item-input", "value"),
    Input("add-btn", "n_clicks"),
    State("new-item-input", "value"),
    prevent_initial_call=True,
)
def add_item(button_clicked, value):
    patched_list = Patch()

    def new_checklist_item():
        return html.Div(
            [
                dcc.Checklist(
                    options=[{"label": "", "value": "done"}],
                    id={"index": button_clicked, "type": "done"},
                    style={"display": "inline"},
                    labelStyle={"display": "inline"},
                ),
                html.Div(
                    [value],
                    id={"index": button_clicked, "type": "output-str"},
                    style={"display": "inline", "margin": "10px"},
                ),
            ]
        )

    patched_list.append(new_checklist_item())
    return patched_list, ""


# Callback to update item styling
@callback(
    Output({"index": MATCH, "type": "output-str"}, "style"),
    Input({"index": MATCH, "type": "done"}, "value"),
    prevent_initial_call=True,
```



▶ Todo App Without Partial Updates

Dash Python **> *Pattern-Matching Callbacks***

Products

Dash

Consulting and Training

Pricing

Enterprise Pricing

About Us

Careers

Resources

Blog

Support

Community Support

Graphing Documentation

Join our mailing list

Sign up to stay in the loop with all things Plotly — from Dash Club to product updates, webinars, and more!

SUBSCRIBE

Copyright © 2025 Plotly. All rights reserved.

Terms of Service Privacy Policy