plotly

Star 23,447

Dash Python > ***Component Argument Order***

Component Argument Order

Dash components each have their own set of properties. **html.Div**, for example, has a `children` property for the content to be displayed in the element, an `id` property to reference this component in callbacks, and a `title` property for tooltips:

```
html.Div(children='Hello World', title='Hover to see this', id='my-div')
```

The component's properties can be defined with keyword arguments or positional arguments. Keyword arguments are defined with the property name and an equals (`=`) sign. Positional arguments precede the keyword arguments and are provided without the property name and the equals (`=`) sign. **Learn more about positional vs keyword arguments in Python**

Positional arguments summary

- Omits the property keyword
- Only used for the first one or two properties
- If a component has `children` as a property, then `children` will be the first positional argument
- `dcc.Dropdown`, `dcc.RadioItems`, and `dcc.Checklist` have `options` as the first argument and `value` as the second
- `dcc.Slider` has `min`, `max`, and `step` as the first three arguments
- Use positional arguments to write shorter, terser code as you can omit the property name

Keyword arguments summary

- Includes the property keyword
- The property keyword makes the relationship between the component and the callback more explicit
- Use keyword arguments to write more readable code where the shorter arguments are declared above the longer arguments
- Use keyword arguments to make your code more readable to other developers who might be less familiar with the components

Default Positional Argument

If a component has `children` as a property, then `children` will be the first positional argument.

In this example with `html.Div`, we provide `children` as the first positional argument:

```
html.Div('Dash: A web application framework for your data.', style={
    'textAlign': 'center',
    'color': 'darkgrey'
})
```

You can also provide it with the `children` keyword:

```
html.Div(
    children='Dash: A web application framework for your data.',
    style={'textAlign': 'center', 'color': 'darkgrey'}
)
```

Keyword arguments allow you to change the order in which properties are supplied. This is equivalent:

```
html.Div(
    style={'textAlign': 'center', 'color': 'darkgrey'},
    children='Dash: A web application framework for your data.'
)
```

Rearranging the order of properties can make your code easier to read.

For example, in the code below `children` is supplied first but contains many lines (sometimes hundreds!). The `style` and `id` properties can get "lost" below the children property, making the code more difficult to read.

```
html.Div([
        # ...
        # many lines of code...
    ],
    style={'textAlign': 'center', 'color': 'darkgrey'},
    id='my-div'
)
```

Instead, placing the short properties above `children` makes the code more readable:

```
html.Div(
    id='my-div',
    style={'textAlign': 'center', 'color': 'darkgrey'},
    children=[
        # ...
        # many lines of code...
    ],
)
```

In our documentation, we'll often include the keyword arguments to make readers aware of the properties of the component. It's important to know the properties of Dash components as every property in a component can be the input, output, or state of a callback.

Other Default Positional Arguments

If the component doesn't have `children` as a property, then the component will have a different default positional argument.

Though `children` is the most common positional argument for components, some components have different default positional arguments.

Dropdown, RadioItems, and Checklist

`Dropdown`, `RadioItems`, and `Checklist`, accept `options` as the first positional argument, and `value` as the second positional argument:

```
app.layout = dcc.RadioItems(
    ['New York City', 'Montreal','San Francisco'],
    'Montreal'
)
```

In the above example, the first argument, `['New York City', 'Montreal','San Francisco']`, sets the `options` property on this `dcc.RadioItems` component. The second argument, `'Montreal'`, sets the `value` property.

As with the `html.Div` example above, keyword arguments can be provided if you want to make it explicit in your code which properties these refer to:

```
app.layout = dcc.RadioItems(
    options=['New York City', 'Montreal','San Francisco'],
    value='Montreal'
)
```

RangeSlider and Slider

`dcc.RangeSlider` and `dcc.Slider` accept `min`, `max`, and `step` as the first three positional arguments:

```
dcc.Slider(0, 20, 5, value=10, id='my-slider')
```

The same `dcc.Slider` with keyword arguments:

```
dcc.Slider(min=0, max=20, step=5, value=10, id='my-slider')
```

Find Default Positional Arguments

See the order that arguments can be provided to a component by either checking that component's reference page, or accessing the documentation in a Python console (e.g. `help(dcc.Dropdown)`) or in your Python IDE.

When viewing the component's reference, the properties will be listed in order. For example, here is the `dcc.Dropdown` reference where you can see that `options` and `value` are listed first and second respectively.

Dropdown Properties

> Access this documentation in your Python terminal with:
>
> ```
> >>> help(dash.dcc.Dropdown)
> ```

> Our recommended IDE for writing Dash apps is Dash Enterprise's **Data Science Workspaces**, which has typeahead support for Dash Component Properties. **Find out if your company is using Dash Enterprise**.

`options` (*list of dicts*; optional): An array of options {label: [string|number], value: [string|number]}, an optional disabled field can be used for each option.

`options` is a list of strings | numbers | booleans | dict | list of dicts with keys:

- **`disabled`** (*boolean*; optional): If True, this option is disabled and cannot be selected.
- **`label`** (*list of or a singular dash component, string or number*; required): The option's label.
- **`search`** (*string*; optional): Optional search value for the option, to use if the label is a component or provide a custom search value different from the label. If no search value and the label is a component, the `value` will be used for search.
- **`title`** (*string*; optional): The HTML 'title' attribute for the option. Allows for information on hover. For more information on this attribute, see **https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/title**.
- **`value`** (*string* | *number* | *boolean*; required): The value of the option. This value corresponds to the items specified in the `value` property.

`value` (*string* | *number* | *boolean* | *list of strings* | *numbers* | *booleans*; optional): The value of the input. If `multi` is False (the default) then value is just a string that corresponds to the values provided in the `options` property. If `multi` is True, then multiple values can be selected at once, and `value` is an array of items with values corresponding to those in the `options` prop.

`multi` (*boolean*; default `False`): If True, the user can select multiple values.

`clearable` (*boolean*; default `True`): Whether or not the dropdown is "clearable", that is, whether or not a small "x" appears on the right of the dropdown that removes the selected value.

`searchable` (*boolean*; default `True`): Whether to enable the searching feature or not.

`search_value` (*string*; optional): The value typed in the DropDown for searching.

`placeholder` (*string*; optional): The grey, default text shown when no option is selected.

`disabled` (*boolean*; default `False`): If True, this dropdown is disabled and the selection cannot be changed.

`closeOnSelect` (*boolean*; default `True`): If False, the menu of the dropdown will not close once a value is selected.

`optionHeight` (*number*; default `35`): height of each option. Can be increased when label lengths would wrap around.

`maxHeight` (*number*; default `200`): height of the options dropdown.

`style` (*dict*; optional): Defines CSS styles which will override styles previously set.

`className` (*string*; optional): className of the dropdown element.

`id` (*string*; optional): The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

`persistence` (*boolean | string | number*; optional): Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If `persisted` is truthy and hasn't changed from its previous value, a `value` that the user has changed while using the app will keep that change, as long as the new `value` also matches what was given originally. Used in conjunction with `persistence_type`.

`persisted_props` (*list of values equal to: 'value'*; default `['value']`): Properties whose user interactions will persist after refreshing the component or the page. Since only `value` is allowed this prop can normally be ignored.

`persistence_type` (*a value equal to: 'local', 'session' or 'memory'*; default `'local'`): Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

Dash Python **> *Component Argument Order***