Data Elements

Table

A table based on Quasar's **QTable**↗ component.

rows: list of row objects
columns: list of column objects (defaults to the columns of the first row *since version 2.0.0*)
column_defaults: optional default column properties, *added in version 2.0.0*
row_key: name of the column containing unique data identifying the row (default: "id")
title: title of the table
selection: selection type ("single" or "multiple"; default: *None*)
pagination: a dictionary correlating to a pagination object or number of rows per page (*None* hides the pagination, 0 means "infinite"; default: *None*).
on_select: callback which is invoked when the selection changes
on_pagination_change: callback which is invoked when the pagination changes

If selection is 'single' or 'multiple', then a *selected* property is accessible containing the selected rows.

main.py

```python
from nicegui import ui

columns = [
    {'name': 'name', 'label': 'Name', 'field': 'name', 'required': True, 'align':
    {'name': 'age', 'label': 'Age', 'field': 'age', 'sortable': True},
]
rows = [
    {'name': 'Alice', 'age': 18},
    {'name': 'Bob', 'age': 21},
    {'name': 'Carol'},
]
ui.table(columns=columns, rows=rows, row_key='name')

ui.run()
```

NiceGUI

Name	Age
Alice	18
Bob	21
Carol	

See **more...**↗

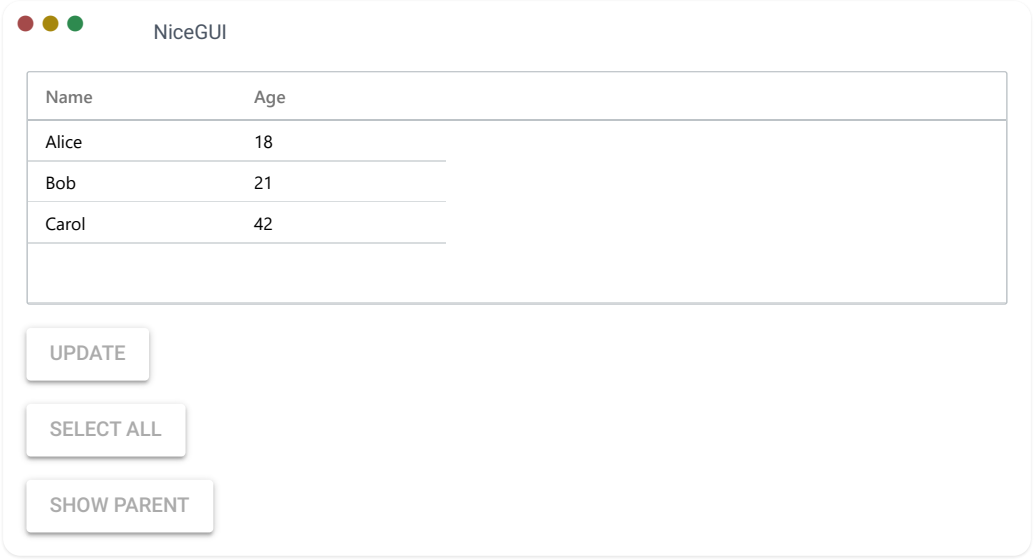
AG Grid

An element to create a grid using **AG Grid**↗.

html_columns: list of columns that should be rendered as HTML (default: `[]`)
theme: AG Grid theme (default: "balham")
auto_size_columns: whether to automatically resize columns to fit the grid width (default: `True`)

main.py

```python
from nicegui import ui

grid = ui.aggrid({
    'defaultColDef': {'flex': 1},
    'columnDefs': [
        {'headerName': 'Name', 'field': 'name'},
        {'headerName': 'Age', 'field': 'age'},
        {'headerName': 'Parent', 'field': 'parent', 'hide': True},
    ],
    'rowData': [
        {'name': 'Alice', 'age': 18, 'parent': 'David'},
        {'name': 'Bob', 'age': 21, 'parent': 'Eve'},
        {'name': 'Carol', 'age': 42, 'parent': 'Frank'},
    ],
    'rowSelection': 'multiple',
}).classes('max-h-40')

def update():
    grid.options['rowData'][0]['age'] += 1
    grid.update()

ui.button('Update', on_click=update)
ui.button('Select all', on_click=lambda: grid.run_grid_method('selectAll'))
ui.button('Show parent', on_click=lambda: grid.run_grid_method('setColumnsVisible'

ui.run()
```



See **more...↗**

Highcharts chart

An element to create a chart using **Highcharts↗**. Updates can be pushed to the chart by changing the *options* property. After data has changed, call the *update* method to refresh the chart.

Due to Highcharts' restrictive license, this element is not part of the standard NiceGUI package. It is maintained in a **separate repository↗** and can be installed with *pip install nicegui[highcharts]*.

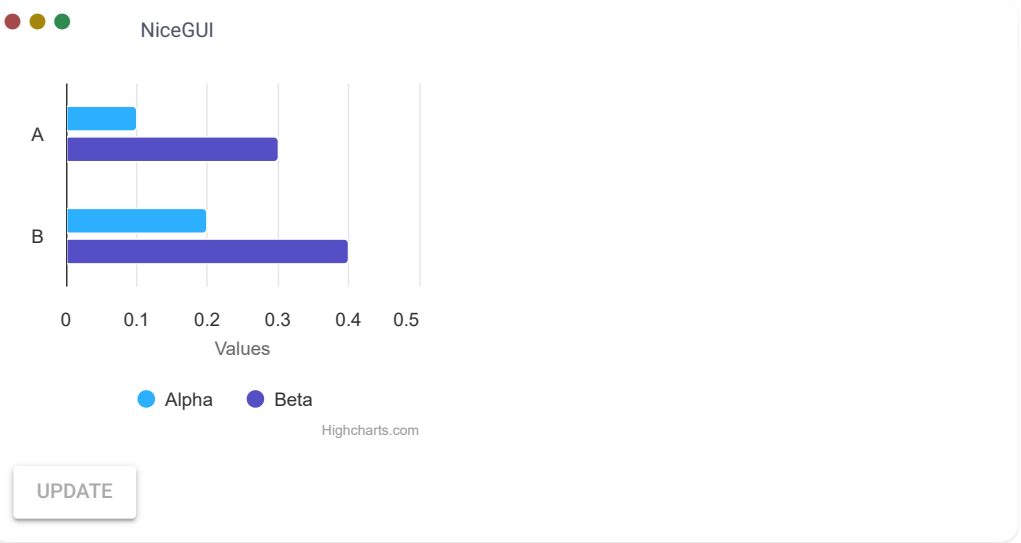
By default, a *Highcharts.chart* is created. To use, e.g., *Highcharts.stockChart* instead, set the *type* property to "stockChart".

options: dictionary of Highcharts options

on_point_drag_start: callback function that is called when a point drag starts
on_point_drag: callback function that is called when a point is dragged
on_point_drop: callback function that is called when a point is dropped

main.py

```python
from nicegui import ui
from random import random

chart = ui.highchart({
    'title': False,
    'chart': {'type': 'bar'},
    'xAxis': {'categories': ['A', 'B']},
    'series': [
        {'name': 'Alpha', 'data': [0.1, 0.2]},
        {'name': 'Beta', 'data': [0.3, 0.4]},
    ],
}).classes('w-full h-64')

def update():
    chart.options['series'][0]['data'][0] = random()
    chart.update()

ui.button('Update', on_click=update)

ui.run()
```



See **more...↗**

Apache EChart

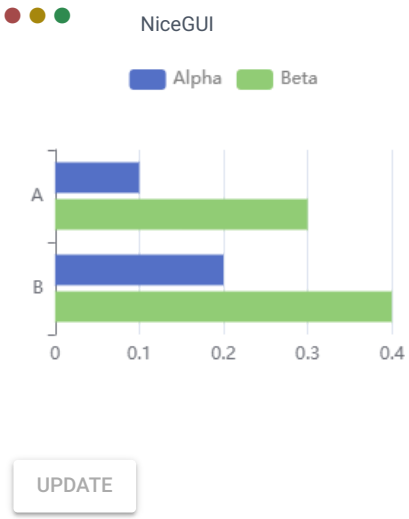
An element to create a chart using **ECharts↗**. Updates can be pushed to the chart by changing the *options* property. After data has changed, call the *update* method to refresh the chart.

options: dictionary of EChart options
on_click_point: callback that is invoked when a point is clicked
enable_3d: enforce importing the echarts-gl library
renderer: renderer to use ("canvas" or "svg", *added in version 2.7.0*)
theme: an EChart theme configuration (dictionary or a URL returning a JSON object, *added in version 2.15.0*)

main.py

```python
from nicegui import ui
from random import random
```

```
    'series': [
        {'type': 'bar', 'name': 'Alpha', 'data': [0.1, 0.2]},
        {'type': 'bar', 'name': 'Beta', 'data': [0.3, 0.4]},
    ],
})

def update():
    echart.options['series'][0]['data'][0] = random()
    echart.update()

ui.button('Update', on_click=update)

ui.run()
```



See **more...↗**

Pyplot Context

Create a context to configure a **Matplotlib↗** plot.

close: whether the figure should be closed after exiting the context; set to *False* if you want to update it later (default: *True*)

kwargs: arguments like *figsize* which should be passed to **pyplot.figure↗**

main.py

```
import numpy as np
from matplotlib import pyplot as plt
from nicegui import ui

with ui.pyplot(figsize=(3, 2)):
    x = np.linspace(0.0, 5.0)
    y = np.cos(2 * np.pi * x) * np.exp(-x)
    plt.plot(x, y, '-')

ui.run()
```

See **more...↗**

Matplotlib

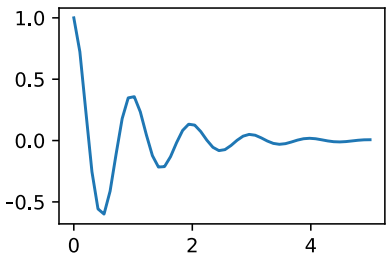
Create a **Matplotlib↗** element rendering a Matplotlib figure. The figure is automatically updated when leaving the figure context.

kwargs: arguments like *figsize* which should be passed to **matplotlib.figure.Figure↗**

main.py

```
import numpy as np
from nicegui import ui

with ui.matplotlib(figsize=(3, 2)).figure as fig:
    x = np.linspace(0.0, 5.0)
    y = np.cos(2 * np.pi * x) * np.exp(-x)
    ax = fig.gca()
    ax.plot(x, y, '-')

ui.run()
```

NiceGUI



See **more...↗**

Line Plot

Create a line plot using pyplot. The *push* method provides live updating when utilized in combination with *ui.timer*.

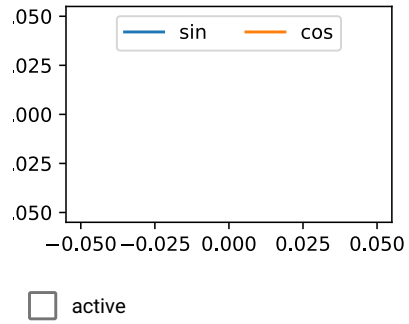
n:	number of lines
limit:	maximum number of datapoints per line (new points will displace the oldest)
update_every:	update plot only after pushing new data multiple times to save CPU and bandwidth
close:	whether the figure should be closed after exiting the context; set to *False* if you want to update it later (default: *True*)
kwargs:	arguments like *figsize* which should be passed to **pyplot.figure↗**

main.py

```
import math
from datetime import datetime
from nicegui import ui

line_plot = ui.line_plot(n=2, limit=20, figsize=(3, 2), update_every=5) \
    .with_legend(['sin', 'cos'], loc='upper center', ncol=2)

def update_line_plot() -> None:
    now = datetime.now()
    x = now.timestamp()
    y1 = math.sin(x)
```

```
line_checkbox = ui.checkbox('active').bind_value(line_updates, 'active')

ui.run()
```

NiceGUI



See **more...↗**

Plotly Element

Renders a Plotly chart. There are two ways to pass a Plotly figure for rendering, see parameter *figure*:

- Pass a *go.Figure* object, see **https://plotly.com/python/↗**
- Pass a Python *dict* object with keys *data*, *layout*, *config* (optional), see **https://plotly.com/javascript/↗**

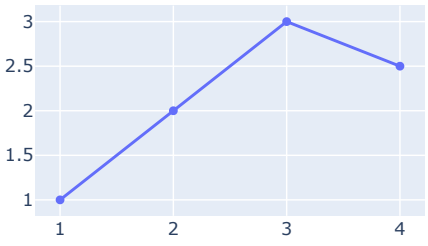
For best performance, use the declarative *dict* approach for creating a Plotly chart.

figure: Plotly figure to be rendered. Can be either a *go.Figure* instance, or a *dict* object with keys *data*, *layout*, *config* (optional).

main.py

```
import plotly.graph_objects as go
from nicegui import ui

fig = go.Figure(go.Scatter(x=[1, 2, 3, 4], y=[1, 2, 3, 2.5]))
fig.update_layout(margin=dict(l=0, r=0, t=0, b=0))
ui.plotly(fig).classes('w-full h-40')

ui.run()
```

NiceGUI



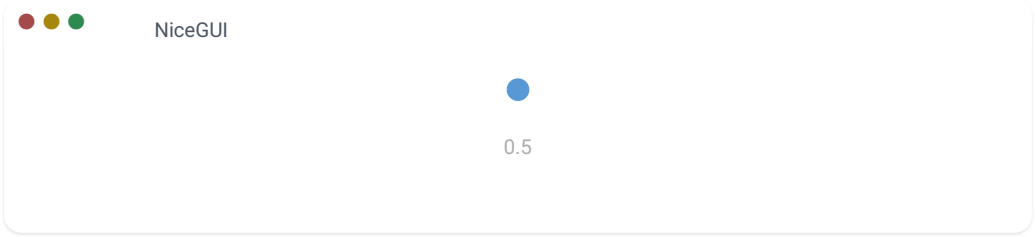
See **more...↗**

Linear Progress

show_value: whether to show a value label in the center (default: *True*)
color: color (either a Quasar, Tailwind, or CSS color or *None*, default: "primary")

main.py

```python
from nicegui import ui

slider = ui.slider(min=0, max=1, step=0.01, value=0.5)
ui.linear_progress().bind_value_from(slider, 'value')

ui.run()
```



See **more...↗**

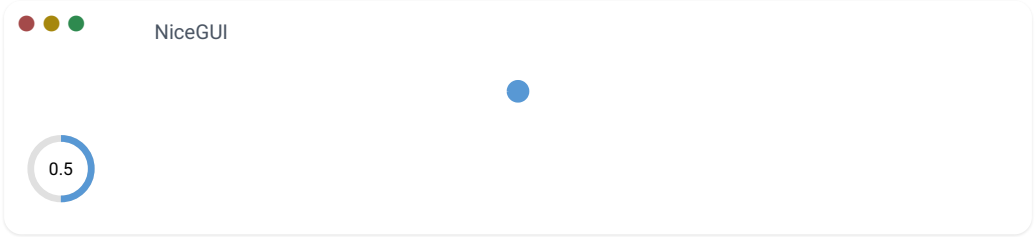
Circular Progress

A circular progress bar wrapping Quasar's **QCircularProgress↗**.

value: the initial value of the field
min: the minimum value (default: 0.0)
max: the maximum value (default: 1.0)
size: the size of the progress circle (default: "xl")
show_value: whether to show a value label in the center (default: *True*)
color: color (either a Quasar, Tailwind, or CSS color or *None*, default: "primary")

main.py

```python
from nicegui import ui

slider = ui.slider(min=0, max=1, step=0.01, value=0.5)
ui.circular_progress().bind_value_from(slider, 'value')

ui.run()
```



See **more...↗**

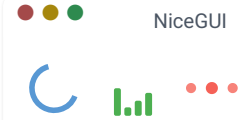
Spinner

This element is based on Quasar's **QSpinner↗** component.

type: type of spinner (e.g. "audio", "ball", "bars", ..., default: "default")
size: size of the spinner (e.g. "3em", "10px", "xl", ..., default: "1em")

main.py

```python
from nicegui import ui

with ui.row():
    ui.spinner(size='lg')
    ui.spinner('audio', size='lg', color='green')
    ui.spinner('dots', size='lg', color='red')

ui.run()
```



See **more...↗**

3D Scene

Display a 3D scene using **three.js↗**. Currently NiceGUI supports boxes, spheres, cylinders/cones, extrusions, straight lines, curves and textured meshes. Objects can be translated, rotated and displayed with different color, opacity or as wireframes. They can also be grouped to apply joint movements.

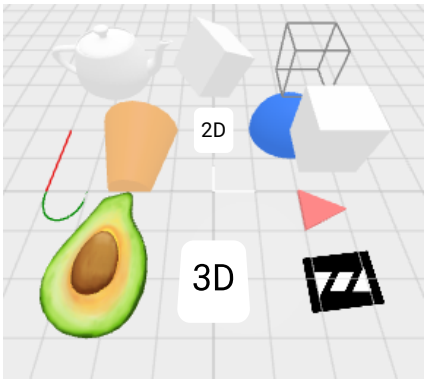
width: width of the canvas
height: height of the canvas
grid: whether to display a grid (boolean or tuple of `size` and `divisions` for **Three.js' GridHelper↗**, default: 100x100)
camera: camera definition, either instance of `ui.scene.perspective_camera` (default) or `ui.scene.orthographic_camera`
on_click: callback to execute when a 3D object is clicked (use `click_events` to specify which events to subscribe to)
click_events: list of JavaScript click events to subscribe to (default: `['click', 'dblclick']`)
on_drag_start: callback to execute when a 3D object is dragged
on_drag_end: callback to execute when a 3D object is dropped
drag_constraints: comma-separated JavaScript expression for constraining positions of dragged objects (e.g. `'x = 0, z = y / 2'`)
background_color: background color of the scene (default: "#eee")

main.py

```python
from nicegui import ui

with ui.scene().classes('w-full h-64') as scene:
    scene.axes_helper()
    scene.sphere().material('#4488ff').move(2, 2)
    scene.cylinder(1, 0.5, 2, 20).material('#ff8800', opacity=0.5).move(-2, 1)
    scene.extrusion([[0, 0], [0, 1], [1, 0.5]], 0.1).material('#ff8888').move(2, -

    with scene.group().move(z=2):
        scene.box().move(x=2)
        scene.box().move(y=2).rotate(0.25, 0.5, 0.75)
        scene.box(wireframe=True).material('#888888').move(x=2, y=2)

    scene.line([-4, 0, 0], [-4, 2, 0]).material('#ff0000')
    scene.curve([-4, 0, 0], [-4, -1, 0], [-3, -1, 0], [-3, 0, 0]).material('#00880

    logo = 'https://avatars.githubusercontent.com/u/2843826'
    scene.texture(logo, [[[0.5, 2, 0], [2.5, 2, 0]],
                         [[0.5, 0, 0], [2.5, 0, 0]]]).move(1, -3)
```

```
    scene.gltf(avocado).scale(40).move(-2, -3, 0.5)

    scene.text('2D', 'background: rgba(0, 0, 0, 0.2); border-radius: 5px; padding:
    scene.text3d('3D', 'background: rgba(0, 0, 0, 0.2); border-radius: 5px; paddin

ui.run()
```

NiceGUI



See **more...**↗

Leaflet map

This element is a wrapper around the **Leaflet**↗ JavaScript library.

center: initial center location of the map (latitude/longitude, default: (0.0, 0.0))
zoom: initial zoom level of the map (default: 13)
draw_control: whether to show the draw toolbar (default: False)
options: additional options passed to the Leaflet map (default: {})
hide_drawn_items: whether to hide drawn items on the map (default: False, *added in version 2.0.0*)
additional_resources: additional resources like CSS or JS files to load (default: None, *added in version 2.11.0*)

main.py

```
from nicegui import ui

m = ui.leaflet(center=(51.505, -0.09))
ui.label().bind_text_from(m, 'center', lambda center: f'Center: {center[0]:.3f}, {
ui.label().bind_text_from(m, 'zoom', lambda zoom: f'Zoom: {zoom}')

with ui.grid(columns=2):
    ui.button('London', on_click=lambda: m.set_center((51.505, -0.090)))
    ui.button('Berlin', on_click=lambda: m.set_center((52.520, 13.405)))
    ui.button(icon='zoom_in', on_click=lambda: m.set_zoom(m.zoom + 1))
    ui.button(icon='zoom_out', on_click=lambda: m.set_zoom(m.zoom - 1))

ui.run()
```

NiceGUI

Tree

Display hierarchical data using Quasar's **QTree**↗ component.

If using IDs, make sure they are unique within the whole tree.

To use checkboxes and `on_tick`, set the `tick_strategy` parameter to "leaf", "leaf-filtered" or "strict".

nodes: hierarchical list of node objects
node_key: property name of each node object that holds its unique id (default: "id")
label_key: property name of each node object that holds its label (default: "label")
children_key: property name of each node object that holds its list of children (default: "children")
on_select: callback which is invoked when the node selection changes
on_expand: callback which is invoked when the node expansion changes
on_tick: callback which is invoked when a node is ticked or unticked
tick_strategy: whether and how to use checkboxes ("leaf", "leaf-filtered" or "strict"; default: None)

main.py

```python
from nicegui import ui

ui.tree([
    {'id': 'numbers', 'children': [{'id': '1'}, {'id': '2'}]},
    {'id': 'letters', 'children': [{'id': 'A'}, {'id': 'B'}]},
], label_key='id', on_select=lambda e: ui.notify(e.value))

ui.run()
```

NiceGUI

See **more...**↗

Log View

Create a log view that allows to add new lines without re-transmitting the whole history to the client.

max_lines: maximum number of lines before dropping oldest ones (default: *None*)

main.py

```python
from datetime import datetime
from nicegui import ui

log = ui.log(max_lines=10).classes('w-full h-20')
ui.button('Log time', on_click=lambda: log.push(datetime.now().strftime('%X.%f')[:

ui.run()
```

NiceGUI

Editor

A WYSIWYG editor based on **Quasar's QEditor**↗. The value is a string containing the formatted text as HTML code.

value: initial value
on_change: callback to be invoked when the value changes

main.py

```
from nicegui import ui

editor = ui.editor(placeholder='Type something here')
ui.markdown().bind_content_from(editor, 'value',
                                backward=lambda v: f'HTML code:\n```\n{v}\n```')

ui.run()
```


See **more...**↗

Code

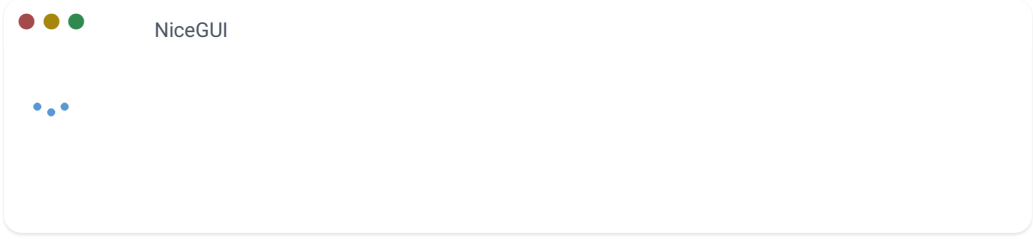
This element displays a code block with syntax highlighting.

In secure environments (HTTPS or localhost), a copy button is displayed to copy the code to the clipboard.

content: code to display
language: language of the code (default: "python")

main.py

```
from nicegui import ui

ui.code('''
    from nicegui import ui

    ui.label('Code inception!')

    ui.run()
''').classes('w-full')

ui.run()
```



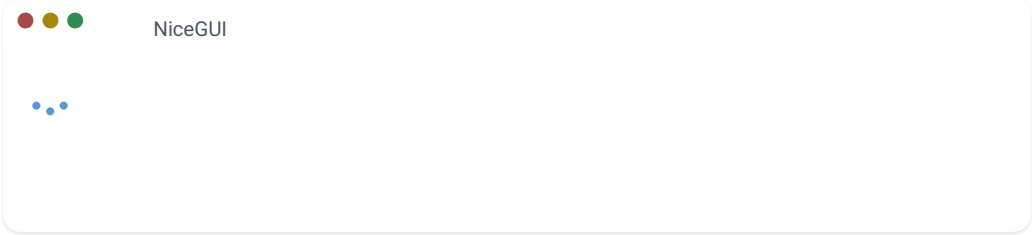
See **more...**↗

An element to create a JSON editor using **JSONEditor**↗. Updates can be pushed to the editor by changing the *properties* property. After data has changed, call the *update* method to refresh the editor.

properties: dictionary of JSONEditor properties
on_select: callback which is invoked when some of the content has been selected
on_change: callback which is invoked when the content has changed
schema: optional **JSON schema**↗ for validating the data being edited (*added in version 2.8.0*)

main.py

```python
from nicegui import ui

json = {
    'array': [1, 2, 3],
    'boolean': True,
    'color': '#82b92c',
    None: None,
    'number': 123,
    'object': {
        'a': 'b',
        'c': 'd',
    },
    'time': 1575599819000,
    'string': 'Hello World',
}
ui.json_editor({'content': {'json': json}},
               on_select=lambda e: ui.notify(f'Select: {e}'),
               on_change=lambda e: ui.notify(f'Change: {e}'))

ui.run()
```



See **more...**↗