plotly package

https://plot.ly/python/ (https://plot.ly/python/)

Plotly's Python API allows users to programmatically access Plotly's server resources.

This package is organized as follows:

Subpackages:

- plotly: all functionality that requires access to Plotly's servers
- graph_objects: objects for designing figures and visualizing data
- matplotlylib: tools to convert matplotlib figures

Modules:

- tools: some helpful tools that do not require access to Plotly's servers
- utils: functions that you probably won't need, but that subpackages use
- version: holds the current API version
- exceptions: defines our custom exception classes

Subpackages

- plotly.graph_objects.choroplethmapbox package (plotly.graph_objects.choroplethmapbox.html)
  - Subpackages (plotly.graph_objects.choroplethmapbox.html#subpackages)
- plotly.graph_objects.cone package (plotly.graph_objects.cone.html)
  - Subpackages (plotly.graph_objects.cone.html#subpackages)
- plotly.graph_objects.contour package (plotly.graph_objects.contour.html)
  - Subpackages (plotly.graph_objects.contour.html#subpackages)
- plotly.graph_objects.contourcarpet package (plotly.graph_objects.contourcarpet.html)
  - Subpackages (plotly.graph_objects.contourcarpet.html#subpackages)
- plotly.graph_objects.densitymap package (plotly.graph_objects.densitymap.html)
  - Subpackages (plotly.graph_objects.densitymap.html#subpackages)
- plotly.graph_objects.densitymapbox package (plotly.graph_objects.densitymapbox.html)
  - Subpackages (plotly.graph_objects.densitymapbox.html#subpackages)
- plotly.graph_objects.funnel package (plotly.graph_objects.funnel.html)
  - Subpackages (plotly.graph_objects.funnel.html#subpackages)
- plotly.graph_objects.funnelarea package (plotly.graph_objects.funnelarea.html)
  - Subpackages (plotly.graph_objects.funnelarea.html#subpackages)
- plotly.graph_objects.heatmap package (plotly.graph_objects.heatmap.html)
  - Subpackages (plotly.graph_objects.heatmap.html#subpackages)
- plotly.graph_objects.heatmapgl package (plotly.graph_objects.heatmapgl.html)
  - Subpackages (plotly.graph_objects.heatmapgl.html#subpackages)
- plotly.graph_objects.histogram package (plotly.graph_objects.histogram.html)
  - Subpackages (plotly.graph_objects.histogram.html#subpackages)
- plotly.graph_objects.histogram2d package (plotly.graph_objects.histogram2d.html)
  - Subpackages (plotly.graph_objects.histogram2d.html#subpackages)
- plotly.graph_objects.histogram2dcontour package (plotly.graph_objects.histogram2dcontour.html)
  - Subpackages (plotly.graph_objects.histogram2dcontour.html#subpackages)
- plotly.graph_objects.icicle package (plotly.graph_objects.icicle.html)
  - Subpackages (plotly.graph_objects.icicle.html#subpackages)
- plotly.graph_objects.image package (plotly.graph_objects.image.html)
  - Subpackages (plotly.graph_objects.image.html#subpackages)
- plotly.graph_objects.indicator package (plotly.graph_objects.indicator.html)
  - Subpackages (plotly.graph_objects.indicator.html#subpackages)
- plotly.graph_objects.isosurface package (plotly.graph_objects.isosurface.html)

Submodules

plotly.animation module

class plotly.animation. **DurationValidator** (*plotly_name='duration'*)

Bases: **_plotly_utils.basevalidators.NumberValidator**

class plotly.animation. **EasingValidator** (*plotly_name='easing'*, *parent_name='batch_animate'*, ***_*)

Bases: **_plotly_utils.basevalidators.EnumeratedValidator**

plotly.basedatatypes module

class plotly.basedatatypes. **BaseFigure** (*data=None*, *layout_plotly=None*, *frames=None*, *skip_invalid=False*, ***kwargs*)

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

Base class for all figure types (both widget and non-widget)

add_hline (*y*, *row='all'*, *col='all'*, *exclude_empty_subplots=True*, *annotation=None*, ***kwargs*)

Add a horizontal line to a plot or subplot that extends infinitely in the x-dimension.

Parameters:

- **y** (*float* (https://docs.python.org/3/library/functions.html#float) *or int* (https://docs.python.org/3/library/functions.html#int)) – A number representing the y coordinate of the horizontal line.
- **exclude_empty_subplots** (*Boolean*) – If True (default) do not place the shape on subplots that have no data plotted on them.
- **row** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot row for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **col** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot column for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **annotation** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or plotly.graph_objects.layout.Annotation. If dict(),*) – it is interpreted as describing an annotation. The annotation is placed relative to the shape based on annotation_position (see below) unless its x or y value has been specified for the annotation passed here. xref and yref are always the same as for the added shape and cannot be overridden.

- **annotation_position** (*a string containing optionally ["top", "bottom"]*) – and [“left”, “right”] specifying where the text should be anchored to on the line. Example positions are “bottom left”, “right top”, “right”, “bottom”. If an annotation is added but annotation_position is not specified, this defaults to “top right”.
- **annotation_*** (*any parameters to go.layout.Annotation can be passed as*) – keywords by prefixing them with “annotation_”. For example, to specify the annotation text “example” you can pass annotation_text=”example” as a keyword argument.
- ****kwargs** – Any named function parameters that can be passed to ‘add_shape’, except for x0, x1, y0, y1 or type.

`add_hrect` (*y0*, *y1*, *row='all'*, *col='all'*, *exclude_empty_subplots=True*, *annotation=None*, ***kwargs*)

Add a rectangle to a plot or subplot that extends infinitely in the x-dimension.

Parameters:

- **y0** (*float* (https://docs.python.org/3/library/functions.html#float) *or int* (https://docs.python.org/3/library/functions.html#int)) – A number representing the y coordinate of one side of the rectangle.
- **y1** (*float* (https://docs.python.org/3/library/functions.html#float) *or int* (https://docs.python.org/3/library/functions.html#int)) – A number representing the y coordinate of the other side of the rectangle.
- **exclude_empty_subplots** (*Boolean*) – If True (default) do not place the shape on subplots that have no data plotted on them.
- **row** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot row for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **col** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot column for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **annotation** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or plotly.graph_objects.layout.Annotation. If dict(),*) – it is interpreted as describing an annotation. The annotation is placed relative to the shape based on annotation_position (see below) unless its x or y value has been specified for the annotation passed here. xref and yref are always the same as for the added shape and cannot be overridden.
- **annotation_position** (*a string containing optionally ["inside", "outside"], ["top", "bottom"]*) – and [“left”, “right”] specifying where the text should be anchored to on the rectangle. Example positions are “outside top left”, “inside bottom”, “right”, “inside left”, “inside” (“outside” is not supported). If an annotation is added but annotation_position is not specified this defaults to “inside top right”.

- **annotation_*** (*any parameters to go.layout.Annotation can be passed as*) – keywords by prefixing them with “annotation_”. For example, to specify the annotation text “example” you can pass annotation_text=”example” as a keyword argument.
- ****kwargs** – Any named function parameters that can be passed to ‘add_shape’, except for x0, x1, y0, y1 or type.

`add_trace` (*trace*, *row=None*, *col=None*, *secondary_y=None*, *exclude_empty_subplots=False*)

Add a trace to the figure

Parameters:

- **trace** (*BaseTraceType or dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Either:
  - An instances of a trace classe from the plotly.graph_objects package (e.g plotly.graph_objects.Scatter, plotly.graph_objects.Bar)
  - or a dicts where:
    - The ‘type’ property specifies the trace type (e.g. ‘scatter’, ‘bar’, ‘area’, etc.). If the dict has no ‘type’ property then ‘scatter’ is assumed.
    - All remaining properties are passed to the constructor of the specified trace type.
- **row** (*'all', int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default)*) – Subplot row index (starting from 1) for the trace to be added. Only valid if figure was created using `plotly.tools.make_subplots` . If ‘all’, addresses all rows in the specified column(s).
- **col** (*'all', int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default)*) – Subplot col index (starting from 1) for the trace to be added. Only valid if figure was created using `plotly.tools.make_subplots` . If ‘all’, addresses all columns in the specified row(s).
- **secondary_y** (*boolean or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – If True, associate this trace with the secondary y-axis of the subplot at the specified row and col. Only valid if all of the following conditions are satisfied:

- The figure was created using `plotly.subplots.make_subplots`.
- The row and col arguments are not None
- The subplot at the specified row and col has type xy (which is the default) and secondary_y True. These properties are specified in the specs argument to make_subplots. See the make_subplots docstring for more info.
- The trace argument is a 2D cartesian trace (scatter, bar, etc.)

- **exclude_empty_subplots** (*boolean*) – If True, the trace will not be added to subplots that don't already have traces.

Returns: The Figure that add_trace was called on
Return type: BaseFigure (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure)

Examples

```
>>> from plotly import subplots
>>> import plotly.graph_objects as go
```

Add two Scatter traces to a figure

```
>>> fig = go.Figure()
>>> fig.add_trace(go.Scatter(x=[1,2,3], y=[2,1,2]))
Figure(...)
>>> fig.add_trace(go.Scatter(x=[1,2,3], y=[2,1,2]))
Figure(...)
```

Add two Scatter traces to vertically stacked subplots

```
>>> fig = subplots.make_subplots(rows=2)
>>> fig.add_trace(go.Scatter(x=[1,2,3], y=[2,1,2]), row=1, col=1)
Figure(...)
>>> fig.add_trace(go.Scatter(x=[1,2,3], y=[2,1,2]), row=2, col=1)
Figure(...)
```

`add_traces`(*data*, *rows=None*, *cols=None*, *secondary_ys=None*, *exclude_empty_subplots=False*)

Add traces to the figure

Parameters:

- **data** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[BaseTraceType or dict* (https://docs.python.org/3/library/stdtypes.html#dict)*]*) –
  A list of trace specifications to be added. Trace specifications may be either:

- Instances of trace classes from the plotly.graph_objects package (e.g plotly.graph_objects.Scatter, plotly.graph_objects.Bar)
- Dicts where:
  - The 'type' property specifies the trace type (e.g. 'scatter', 'bar', 'area', etc.). If the dict has no 'type' property then 'scatter' is assumed.
  - All remaining properties are passed to the constructor of the specified trace type.

- **rows** (*None* (https://docs.python.org/3/library/constants.html#None)*, list* (https://docs.python.org/3/library/stdtypes.html#list)*[int* (https://docs.python.org/3/library/functions.html#int)*], or int* (https://docs.python.org/3/library/functions.html#int) *(default None)*) – List of subplot row indexes (starting from 1) for the traces to be added. Only valid if figure was created using `plotly.tools.make_subplots` If a single integer is passed, all traces will be added to row number
- **cols** (*None* (https://docs.python.org/3/library/constants.html#None) *or list* (https://docs.python.org/3/library/stdtypes.html#list)*[int* (https://docs.python.org/3/library/functions.html#int)*] (default None)*) – List of subplot column indexes (starting from 1) for the traces to be added. Only valid if figure was created using `plotly.tools.make_subplots` If a single integer is passed, all traces will be added to column number

secondary_ys: None or list[boolean] (default None)

List of secondary_y booleans for traces to be added. See the docstring for `add_trace` for more info.

exclude_empty_subplots: boolean

If True, the trace will not be added to subplots that don't already have traces.

Returns:	The Figure that add_traces was called on
Return type:	BaseFigure (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure)

Examples

```
>>> from plotly import subplots
>>> import plotly.graph_objects as go
```

Add two Scatter traces to a figure

```
>>> fig = go.Figure()
>>> fig.add_traces([go.Scatter(x=[1,2,3], y=[2,1,2]),
...                 go.Scatter(x=[1,2,3], y=[2,1,2])])
Figure(...)
```

Add two Scatter traces to vertically stacked subplots

```
>>> fig = subplots.make_subplots(rows=2)
>>> fig.add_traces([go.Scatter(x=[1,2,3], y=[2,1,2]),
...                 go.Scatter(x=[1,2,3], y=[2,1,2])],
...                 rows=[1, 2], cols=[1, 1])
Figure(...)
```

add_vline (*x*, *row='all'*, *col='all'*, *exclude_empty_subplots=True*, *annotation=None*, ***kwargs*)

Add a vertical line to a plot or subplot that extends infinitely in the y-dimension.

Parameters:

- **x** (*float* (https://docs.python.org/3/library/functions.html#float) *or int* (https://docs.python.org/3/library/functions.html#int)) – A number representing the x coordinate of the vertical line.
- **exclude_empty_subplots** (*Boolean*) – If True (default) do not place the shape on subplots that have no data plotted on them.
- **row** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot row for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **col** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot column for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **annotation** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or plotly.graph_objects.layout.Annotation. If dict(),*) – it is interpreted as describing an annotation. The annotation is placed relative to the shape based on annotation_position (see below) unless its x or y value has been specified for the annotation passed here. xref and yref are always the same as for the added shape and cannot be overridden.
- **annotation_position** (*a string containing optionally ["top", "bottom"]*) – and [“left”, “right”] specifying where the text should be anchored to on the line. Example positions are “bottom left”, “right top”, “right”, “bottom”. If an annotation is added but annotation_position is not specified, this defaults to “top right”.
- **annotation_*** (*any parameters to go.layout.Annotation can be passed as*) – keywords by prefixing them with “annotation_”. For example, to specify the annotation text “example” you can pass annotation_text=”example” as a

keyword argument.

- ****kwargs** – Any named function parameters that can be passed to ‘add_shape’, except for x0, x1, y0, y1 or type.

add_vrect(*x0*, *x1*, *row='all'*, *col='all'*, *exclude_empty_subplots=True*, *annotation=None*, ***kwargs*)

Add a rectangle to a plot or subplot that extends infinitely in the y-dimension.

Parameters:

- **x0** (*float* (https://docs.python.org/3/library/functions.html#float) *or int* (https://docs.python.org/3/library/functions.html#int)) – A number representing the x coordinate of one side of the rectangle.
- **x1** (*float* (https://docs.python.org/3/library/functions.html#float) *or int* (https://docs.python.org/3/library/functions.html#int)) – A number representing the x coordinate of the other side of the rectangle.
- **exclude_empty_subplots** (*Boolean*) – If True (default) do not place the shape on subplots that have no data plotted on them.
- **row** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot row for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **col** (*None* (https://docs.python.org/3/library/constants.html#None)*, int* (https://docs.python.org/3/library/functions.html#int) *or 'all'*) – Subplot column for shape indexed starting at 1. If ‘all’, addresses all rows in the specified column(s). If both row and col are None, addresses the first subplot if subplots exist, or the only plot. By default is “all”.
- **annotation** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or plotly.graph_objects.layout.Annotation. If dict(),*) – it is interpreted as describing an annotation. The annotation is placed relative to the shape based on annotation_position (see below) unless its x or y value has been specified for the annotation passed here. xref and yref are always the same as for the added shape and cannot be overridden.
- **annotation_position** (*a string containing optionally ["inside", "outside"], ["top", "bottom"]*) – and [“left”, “right”] specifying where the text should be anchored to on the rectangle. Example positions are “outside top left”, “inside bottom”, “right”, “inside left”, “inside” (“outside” is not supported). If an annotation is added but annotation_position is not specified this defaults to “inside top right”.
- **annotation_*** (*any parameters to go.layout.Annotation can be passed as*) – keywords by prefixing them with “annotation_”. For example, to specify the annotation text “example” you can pass annotation_text=”example” as a keyword argument.
- ****kwargs** – Any named function parameters that can be passed to ‘add_shape’, except for x0, x1, y0, y1 or type.

`append_trace` (*trace*, *row*, *col*) ¶

Add a trace to the figure bound to axes at the specified row, col index.

A row, col index grid is generated for figures created with plotly.tools.make_subplots, and can be viewed with the `print_grid` method

Parameters:

- **trace** – The data trace to be bound
- **row** (*int* (https://docs.python.org/3/library/functions.html#int)) – Subplot row index (see Figure.print_grid)
- **col** (*int* (https://docs.python.org/3/library/functions.html#int)) – Subplot column index (see Figure.print_grid)

Examples

```
>>> from plotly import tools
>>> import plotly.graph_objects as go
>>> # stack two subplots vertically
>>> fig = tools.make_subplots(rows=2)
```

This is the format of your plot grid: [(1,1) x1,y1] [(2,1) x2,y2]

```
>>> fig.append_trace(go.Scatter(x=[1,2,3], y=[2,1,2]), row=1, col=1)
>>> fig.append_trace(go.Scatter(x=[1,2,3], y=[2,1,2]), row=2, col=1)
```

`batch_animate` (*duration=500*, *easing='cubic-in-out'*)

Context manager to animate trace / layout updates

Parameters:

- **duration** (*number*) – The duration of the transition, in milliseconds. If equal to zero, updates are synchronous.
- **easing** (*string*) –
  The easing function used for the transition. One of:

- linear
- quad
- cubic
- sin
- exp
- circle
- elastic
- back
- bounce
- linear-in
- quad-in
- cubic-in
- sin-in
- exp-in
- circle-in
- elastic-in
- back-in
- bounce-in
- linear-out
- quad-out
- cubic-out
- sin-out
- exp-out
- circle-out
- elastic-out
- back-out
- bounce-out
- linear-in-out
- quad-in-out
- cubic-in-out
- sin-in-out
- exp-in-out
- circle-in-out
- elastic-in-out
- back-in-out

- bounce-in-out

Examples

Suppose we have a figure widget, `fig`, with a single trace.

```
>>> import plotly.graph_objects as go
>>> fig = go.FigureWidget(data=[{'y': [3, 4, 2]}])
```

1) Animate a change in the xaxis and yaxis ranges using default duration and easing parameters.

```
>>> with fig.batch_animate():
...     fig.layout.xaxis.range = [0, 5]
...     fig.layout.yaxis.range = [0, 10]
```

2) Animate a change in the size and color of the trace's markers over 2 seconds using the elastic-in-out easing method

```
>>> with fig.batch_animate(duration=2000, easing='elastic-in-out'):
...     fig.data[0].marker.color = 'green'
...     fig.data[0].marker.size = 20
```

`batch_update` ()

A context manager that batches up trace and layout assignment operations into a singe plotly_update message that is executed when the context exits.

Examples

For example, suppose we have a figure widget, `fig`, with a single trace.

```
>>> import plotly.graph_objects as go
>>> fig = go.FigureWidget(data=[{'y': [3, 4, 2]}])
```

If we want to update the xaxis range, the yaxis range, and the marker color, we could do so using a series of three property assignments as follows:

```
>>> fig.layout.xaxis.range = [0, 5]
>>> fig.layout.yaxis.range = [0, 10]
>>> fig.data[0].marker.color = 'green'
```

This will work, however it will result in three messages being sent to the front end (two relayout messages for the axis range updates followed by one restyle message for the marker color update). This can cause the plot to appear to stutter as the three updates are applied incrementally.

We can avoid this problem by performing these three assignments in a `batch_update` context as follows:

```
>>> with fig.batch_update():
...     fig.layout.xaxis.range = [0, 5]
...     fig.layout.yaxis.range = [0, 10]
...     fig.data[0].marker.color = 'green'
```

Now, these three property updates will be sent to the frontend in a single update message, and they will be applied by the front end simultaneously.

property **data**

The `data` property is a tuple of the figure's trace objects

Returns:

Return type: tuple (https://docs.python.org/3/library/stdtypes.html#tuple)[BaseTraceType]

for_each_trace (*fn*, *selector=None*, *row=None*, *col=None*, *secondary_y=None*)

Apply a function to all traces that satisfy the specified selection criteria

Parameters:

- **fn** – Function that inputs a single trace object.
- **selector** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)*, function, int* (https://docs.python.org/3/library/functions.html#int)*, str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Dict to use as selection criteria. Traces will be selected if they contain properties corresponding to all of the dictionary's keys, with values that exactly match the supplied values. If None (the default), all traces are selected. If a function, it must be a function accepting a single argument and returning a boolean. The function will be called on each trace and those for which the function returned True will be in the selection. If an int N, the Nth trace matching row and col will be selected (N can be negative). If a string S, the selector is equivalent to dict(type=S).
- **row** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Subplot row and column index of traces to select. To select traces by row and column, the Figure must have been created using plotly.subplots.make_subplots. If None (the default), all traces are selected.
- **col** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Subplot row and column index of traces to select. To select traces by row and column, the Figure must have been created using plotly.subplots.make_subplots. If None (the default), all traces are selected.
- **secondary_y** (*boolean or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) –
  - If True, only select traces associated with the secondary y-axis of the subplot.
  - If False, only select traces associated with the primary y-axis of the subplot.

- If None (the default), do not filter traces based on secondary y-axis.

To select traces by secondary y-axis, the Figure must have been created using plotly.subplots.make_subplots. See the docstring for the specs argument to make_subplots for more info on creating subplots with secondary y-axes.

Returns: Returns the Figure object that the method was called on

Return type: self

property **frames**

The `frames` property is a tuple of the figure's frame objects

Returns:

Return type: tuple (https://docs.python.org/3/library/stdtypes.html#tuple) [plotly.graph_objects.Frame (plotly.graph_objects.html#plotly.graph_objects.Frame)]

full_figure_for_development(*warn=True*, *as_dict=False*)

Compute default values for all attributes not specified in the input figure and returns the output as a "full" figure. This function calls Plotly.js via Kaleido to populate unspecified attributes. This function is intended for interactive use during development to learn more about how Plotly.js computes default values and is not generally necessary or recommended for production use.

Parameters:

- **fig** – Figure object or dict representing a figure
- **warn** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If False, suppress warnings about not using this in production.
- **as_dict** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If True, output is a dict with some keys that go.Figure can't parse. If False, output is a go.Figure with unparseable keys skipped.

Returns: The full figure

Return type: plotly.graph_objects.Figure (plotly.graph_objects.html#plotly.graph_objects.Figure) or dict (https://docs.python.org/3/library/stdtypes.html#dict)

get_subplot(*row*, *col*, *secondary_y=False*)

Return an object representing the subplot at the specified row and column. May only be used on Figures created using plotly.tools.make_subplots

Parameters:

- **row** (*int* (https://docs.python.org/3/library/functions.html#int)) – 1-based index of subplot row
- **col** (*int* (https://docs.python.org/3/library/functions.html#int)) – 1-based index of subplot column
- **secondary_y** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If True, select the subplot that consists of the x-axis and the secondary y-axis at the specified row/col. Only valid if the subplot at row/col is an 2D cartesian subplot that was created with a secondary y-axis. See the docstring for the specs argument to make_subplots for more info on creating a subplot with a secondary y-axis.

Returns:

- None: if subplot is empty
- plotly.graph_objects.layout.Scene: if subplot type is 'scene'
- plotly.graph_objects.layout.Polar: if subplot type is 'polar'
- plotly.graph_objects.layout.Ternary: if subplot type is 'ternary'
- plotly.graph_objects.layout.Mapbox: if subplot type is 'ternary'
- SubplotDomain namedtuple with `x` and `y` fields: if subplot type is 'domain'.
  - x: length 2 list of the subplot start and stop width
  - y: length 2 list of the subplot start and stop height
- SubplotXY namedtuple with `xaxis` and `yaxis` fields: if subplot type is 'xy'.
  - xaxis: plotly.graph_objects.layout.XAxis instance for subplot
  - yaxis: plotly.graph_objects.layout.YAxis instance for subplot

Return type: subplot

property **`layout`**

The `layout` property of the figure

Returns:

Return type: plotly.graph_objects.Layout (plotly.graph_objects.Layout.html#plotly.graph_objects.Layout)

`plotly_relayout`(*relayout_data*, ***kwargs*)

Perform a Plotly relayout operation on the figure's layout

Parameters: **relayout_data** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Dict of layout updates
dict keys are strings that specify the properties to be updated. Nested properties are expressed by joining successive keys on '.' characters (e.g. 'xaxis.range')
dict values are the values to use to update the layout.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

`plotly_restyle`(*restyle_data*, *trace_indexes=None*, ***kwargs*)

Perform a Plotly restyle operation on the figure's traces

Parameters:

- **restyle_data** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Dict of trace style updates.
  Keys are strings that specify the properties to be updated. Nested properties are expressed by joining successive keys on '.' characters (e.g. 'marker.color').
  Values may be scalars or lists. When values are scalars, that scalar value is applied to all traces specified by the `trace_indexes` parameter. When values are lists, the restyle operation will cycle through the elements of the list as it

cycles through the traces specified by the `trace_indexes` parameter. Caution: To use plotly_restyle to update a list property (e.g. the `x` property of the scatter trace), the property value should be a scalar list containing the list to update with. For example, the following command would be used to update the 'x' property of the first trace to the list [1, 2, 3]

```
>>> import plotly.graph_objects as go
>>> fig = go.Figure(go.Scatter(x=[2, 4, 6]))
>>> fig.plotly_restyle({'x': [[1, 2, 3]]}, 0)
```

- **trace_indexes** (*int* (https://docs.python.org/3/library/functions.html#int) *or list of int*) – Trace index, or list of trace indexes, that the restyle operation applies to. Defaults to all trace indexes.

Returns:
Return type: None (https://docs.python.org/3/library/constants.html#None)

`plotly_update` (*restyle_data=None*, *relayout_data=None*, *trace_indexes=None*, ***kwargs*)

Perform a Plotly update operation on the figure.

Note: This operation both mutates and returns the figure

Parameters:
- **restyle_data** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Traces update specification. See the docstring for the `plotly_restyle` method for details
- **relayout_data** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Layout update specification. See the docstring for the `plotly_relayout` method for details
- **trace_indexes** – Trace index, or list of trace indexes, that the update operation applies to. Defaults to all trace indexes.

Returns: None
Return type: BaseFigure (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure)

`pop` (*key*, **args*)

Remove the value associated with the specified key and return it

Parameters:
- **key** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Property name
- **dflt** – The default value to return if key was not found in figure

Returns: The removed value that was previously associated with key
Return type: value
Raises: **KeyError** (https://docs.python.org/3/library/exceptions.html#KeyError) – If key is not in object and no dflt argument specified

`print_grid` ()

Print a visual layout of the figure's axes arrangement. This is only valid for figures that are created with plotly.tools.make_subplots.

`select_traces` (*selector=None*, *row=None*, *col=None*, *secondary_y=None*)

Select traces from a particular subplot cell and/or traces that satisfy custom selection criteria.

Parameters:

- **selector** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)*, function, int* (https://docs.python.org/3/library/functions.html#int)*, str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Dict to use as selection criteria. Traces will be selected if they contain properties corresponding to all of the dictionary's keys, with values that exactly match the supplied values. If None (the default), all traces are selected. If a function, it must be a function accepting a single argument and returning a boolean. The function will be called on each trace and those for which the function returned True will be in the selection. If an int N, the Nth trace matching row and col will be selected (N can be negative). If a string S, the selector is equivalent to dict(type=S).
- **row** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Subplot row and column index of traces to select. To select traces by row and column, the Figure must have been created using plotly.subplots.make_subplots. If None (the default), all traces are selected.
- **col** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Subplot row and column index of traces to select. To select traces by row and column, the Figure must have been created using plotly.subplots.make_subplots. If None (the default), all traces are selected.
- **secondary_y** (*boolean or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) –
  - If True, only select traces associated with the secondary y-axis of the subplot.
  - If False, only select traces associated with the primary y-axis of the subplot.
  - If None (the default), do not filter traces based on secondary y-axis.

  To select traces by secondary y-axis, the Figure must have been created using plotly.subplots.make_subplots. See the docstring for the specs argument to make_subplots for more info on creating subplots with secondary y-axes.

Returns: Generator that iterates through all of the traces that satisfy all of the specified selection criteria

Return type: generator

`set_subplots` (*rows=None*, *cols=None*, ***make_subplots_args*)

Add subplots to this figure. If the figure already contains subplots, then this throws an error. Accepts any keyword arguments that plotly.subplots.make_subplots accepts.

`show` (**args*, ***kwargs*)

Show a figure using either the default renderer(s) or the renderer(s) specified by the renderer argument

Parameters:

- **renderer** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – A string containing the names of one or more registered renderers (separated by '+' characters) or None. If None, then the default renderers specified in plotly.io.renderers.default are used.
- **validate** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if the figure should be validated before being shown, False otherwise.
- **width** (*int* (https://docs.python.org/3/library/functions.html#int) *or float* (https://docs.python.org/3/library/functions.html#float)) – An integer or float that determines the number of pixels wide the plot is. The default is set in plotly.js.
- **height** (*int* (https://docs.python.org/3/library/functions.html#int) *or float* (https://docs.python.org/3/library/functions.html#float)) – An integer or float that determines the number of pixels wide the plot is. The default is set in plotly.js.
- **config** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – A dict of parameters to configure the figure. The defaults are set in plotly.js.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

`to_dict` ()

Convert figure to a dictionary

Note: the dictionary includes the properties explicitly set by the user, it does not include default values of unspecified properties

Returns:

Return type: dict (https://docs.python.org/3/library/stdtypes.html#dict)

`to_html` (**args*, ***kwargs*)

Convert a figure to an HTML string representation.

Parameters:

- **config** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Plotly.js figure config options
- **auto_play** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default=True)*) – Whether to automatically start the animation sequence on page load if the figure contains frames. Has no effect if the figure does not contain frames.
- **include_plotlyjs** (*bool* (https://docs.python.org/3/library/functions.html#bool) *or string (default True)*) –
  Specifies how the plotly.js library is included/loaded in the output div string.
  If True, a script tag containing the plotly.js source code (~3MB) is included in the output. HTML files generated with this option are fully self-contained and can be used offline.

If ‘cdn’, a script tag that references the plotly.js CDN is included in the output. HTML files generated with this option are about 3MB smaller than those generated with include_plotlyjs=True, but they require an active internet connection in order to load the plotly.js library.
If ‘directory’, a script tag is included that references an external plotly.min.js bundle that is assumed to reside in the same directory as the HTML file.
If ‘require’, Plotly.js is loaded using require.js. This option assumes that require.js is globally available and that it has been globally configured to know how to find Plotly.js as ‘plotly’. This option is not advised when full_html=True as it will result in a non-functional html file.
If a string that ends in ‘.js’, a script tag is included that references the specified path. This approach can be used to point the resulting HTML file to an alternative CDN or local bundle.
If False, no script tag referencing plotly.js is included. This is useful when the resulting div string will be placed inside an HTML document that already loads plotly.js. This option is not advised when full_html=True as it will result in a non-functional html file.

- **include_mathjax** (*bool* (https://docs.python.org/3/library/functions.html#bool) *or string (default False)*) –
  Specifies how the MathJax.js library is included in the output html div string. MathJax is required in order to display labels with LaTeX typesetting.
  If False, no script tag referencing MathJax.js will be included in the output.
  If ‘cdn’, a script tag that references a MathJax CDN location will be included in the output. HTML div strings generated with this option will be able to display LaTeX typesetting as long as internet access is available.
  If a string that ends in ‘.js’, a script tag is included that references the specified path. This approach can be used to point the resulting HTML div string to an alternative CDN.

- **post_script** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or list* (https://docs.python.org/3/library/stdtypes.html#list) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – JavaScript snippet(s) to be included in the resulting div just after plot creation. The string(s) may include ‘{plot_id}’ placeholders that will then be replaced by the `id` of the div element that the plotly.js figure is associated with. One application for this script is to install custom plotly.js event handlers.

- **full_html** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – If True, produce a string containing a complete HTML document starting with an <html> tag. If False, produce a string containing a single element.

- **animation_opts** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – dict of custom animation parameters to be passed to the function Plotly.animate in Plotly.js. See https://github.com/plotly/plotly.js/blob/master/src/plots/animation_attributes.js (https://github.com/plotly/plotly.js/blob/master/src/plots/animation_attributes.js) for available options. Has no effect if the figure does not contain frames, or auto_play is False.

- **default_width** (*number or str* (https://docs.python.org/3/library/stdtypes.html#str) *(default '100%')*) – The default figure width/height to use if the provided figure does not specify its own layout.width/layout.height property. May be specified in pixels as an integer (e.g. 500), or as a css width style string (e.g. '500px', '100%').
- **default_height** (*number or str* (https://docs.python.org/3/library/stdtypes.html#str) *(default '100%')*) – The default figure width/height to use if the provided figure does not specify its own layout.width/layout.height property. May be specified in pixels as an integer (e.g. 500), or as a css width style string (e.g. '500px', '100%').
- **validate** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if the figure should be validated before being converted to JSON, False otherwise.
- **div_id** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *(default None)*) – If provided, this is the value of the id attribute of the div tag. If None, the id attribute is a UUID.

Returns: Representation of figure as an HTML div string

Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

`to_image` (**args*, ***kwargs*)

Convert a figure to a static image bytes string

Parameters:

- **format** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The desired image format. One of
  - 'png'
  - 'jpg' or 'jpeg'
  - 'webp'
  - 'svg'
  - 'pdf'
  - 'eps' (Requires the poppler library to be installed)

  If not specified, will default to `plotly.io.config.default_format`
- **width** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The width of the exported image in layout pixels. If the `scale` property is 1.0, this will also be the width of the exported image in physical pixels.
  If not specified, will default to `plotly.io.config.default_width`
- **height** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The height of the exported image in layout pixels. If the `scale` property is 1.0, this will also be the height of the exported image in physical pixels.
  If not specified, will default to `plotly.io.config.default_height`

- **scale** (*int* (https://docs.python.org/3/library/functions.html#int) *or float* (https://docs.python.org/3/library/functions.html#float) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The scale factor to use when exporting the figure. A scale factor larger than 1.0 will increase the image resolution with respect to the figure's layout pixel dimensions. Whereas as scale factor of less than 1.0 will decrease the image resolution.
  If not specified, will default to `plotly.io.config.default_scale`
- **validate** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True if the figure should be validated before being converted to an image, False otherwise.
- **engine** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) –
  Image export engine to use:
  - "kaleido": Use Kaleido for image export
  - "orca": Use Orca for image export
  - "auto" (default): Use Kaleido if installed, otherwise use orca

Returns: The image data
Return type: bytes (https://docs.python.org/3/library/stdtypes.html#bytes)

`to_json` (**args*, ***kwargs*)

Convert a figure to a JSON string representation

Parameters:
- **validate** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if the figure should be validated before being converted to JSON, False otherwise.
- **pretty** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default False)*) – True if JSON representation should be pretty-printed, False if representation should be as compact as possible.
- **remove_uids** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if trace UIDs should be omitted from the JSON representation
- **engine** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *(default None)*) –
  The JSON encoding engine to use. One of:
  - "json" for an encoder based on the built-in Python json module
  - "orjson" for a fast encoder the requires the orjson package

  If not specified, the default encoder is set to the current value of plotly.io.json.config.default_encoder.

Returns: Representation of figure as a JSON string
Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

`to_ordered_dict` (*skip_uid=True*)

`to_plotly_json` ()

Convert figure to a JSON representation as a Python dict

Note: May include some JSON-invalid data types, use the `PlotlyJSONEncoder` util or the `to_json` method to encode to a string.

Returns:
Return type: dict (https://docs.python.org/3/library/stdtypes.html#dict)

`update` (*dict1=None*, *overwrite=False*, ***kwargs*)

Update the properties of the figure with a dict and/or with keyword arguments.

This recursively updates the structure of the figure object with the values in the input dict / keyword arguments.

Parameters:

- **dict1** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Dictionary of properties to be updated
- **overwrite** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If True, overwrite existing properties. If False, apply updates to existing properties recursively, preserving existing properties that are not specified in the update operation.
- **kwargs** – Keyword/value pair of properties to be updated

Examples

```
>>> import plotly.graph_objects as go
>>> fig = go.Figure(data=[{'y': [1, 2, 3]}])
>>> fig.update(data=[{'y': [4, 5, 6]}])
Figure(...)
>>> fig.to_plotly_json()
    {'data': [{'type': 'scatter',
       'uid': 'e86a7c7a-346a-11e8-8aa8-a0999b0c017b',
       'y': array([4, 5, 6], dtype=int32)}],
     'layout': {}}
```

```
>>> fig = go.Figure(layout={'xaxis':
...                         {'color': 'green',
...                          'range': [0, 1]}})
>>> fig.update({'layout': {'xaxis': {'color': 'pink'}}})
Figure(...)
>>> fig.to_plotly_json()
    {'data': [],
     'layout': {'xaxis':
                {'color': 'pink',
                 'range': [0, 1]}}}
```

Returns: Updated figure

Return type: BaseFigure (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure)

`update_layout` (*dict1=None*, *overwrite=False*, ***kwargs*)

Update the properties of the figure's layout with a dict and/or with keyword arguments.

This recursively updates the structure of the original layout with the values in the input dict / keyword arguments.

Parameters:

- **dict1** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Dictionary of properties to be updated
- **overwrite** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If True, overwrite existing properties. If False, apply updates to existing properties recursively, preserving existing properties that are not specified in the update operation.
- **kwargs** – Keyword/value pair of properties to be updated

Returns: The Figure object that the update_layout method was called on

Return type: BaseFigure (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure)

`update_traces` (*patch=None*, *selector=None*, *row=None*, *col=None*, *secondary_y=None*, *overwrite=False*, ***kwargs*)

Perform a property update operation on all traces that satisfy the specified selection criteria

Parameters:

- **patch** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Dictionary of property updates to be applied to all traces that satisfy the selection criteria.
- **selector** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)*, function, int* (https://docs.python.org/3/library/functions.html#int)*, str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Dict to use as selection criteria. Traces will be selected if they contain properties corresponding to all of the dictionary's keys, with values that exactly match the supplied values. If None (the default), all traces are selected. If a function, it must be a function accepting a single argument and returning a boolean. The function will be called on each trace and those for which the function returned True will be in the selection. If an int N, the Nth trace matching row and col will be selected (N can be negative). If a string S, the selector is equivalent to dict(type=S).
- **row** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Subplot row and column index of traces to select. To select traces by row and column, the Figure must have been created using plotly.subplots.make_subplots. If None (the default), all traces are selected.

- **col** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Subplot row and column index of traces to select. To select traces by row and column, the Figure must have been created using plotly.subplots.make_subplots. If None (the default), all traces are selected.
- **secondary_y** (*boolean or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) –
  - If True, only select traces associated with the secondary y-axis of the subplot.
  - If False, only select traces associated with the primary y-axis of the subplot.
  - If None (the default), do not filter traces based on secondary y-axis.

  To select traces by secondary y-axis, the Figure must have been created using plotly.subplots.make_subplots. See the docstring for the specs argument to make_subplots for more info on creating subplots with secondary y-axes.
- **overwrite** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If True, overwrite existing properties. If False, apply updates to existing properties recursively, preserving existing properties that are not specified in the update operation.
- ****kwargs** – Additional property updates to apply to each selected trace. If a property is specified in both patch and in **kwargs then the one in **kwargs takes precedence.

Returns: Returns the Figure object that the method was called on
Return type: self

`write_html` (**args*, ***kwargs*)

Write a figure to an HTML file representation

Parameters:

- **file** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or writeable*) – A string representing a local file path or a writeable object (e.g. a pathlib.Path object or an open file descriptor)
- **config** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Plotly.js figure config options
- **auto_play** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default=True)*) – Whether to automatically start the animation sequence on page load if the figure contains frames. Has no effect if the figure does not contain frames.
- **include_plotlyjs** (*bool* (https://docs.python.org/3/library/functions.html#bool) *or string (default True)*) –
  Specifies how the plotly.js library is included/loaded in the output div string.
  If True, a script tag containing the plotly.js source code (~3MB) is included in the output. HTML files generated with this option are fully self-contained and can be used offline.

If ‘cdn’, a script tag that references the plotly.js CDN is included in the output. HTML files generated with this option are about 3MB smaller than those generated with include_plotlyjs=True, but they require an active internet connection in order to load the plotly.js library.
If ‘directory’, a script tag is included that references an external plotly.min.js bundle that is assumed to reside in the same directory as the HTML file. If `file` is a string to a local file path and `full_html` is True, then the plotly.min.js bundle is copied into the directory of the resulting HTML file. If a file named plotly.min.js already exists in the output directory then this file is left unmodified and no copy is performed. HTML files generated with this option can be used offline, but they require a copy of the plotly.min.js bundle in the same directory. This option is useful when many figures will be saved as HTML files in the same directory because the plotly.js source code will be included only once per output directory, rather than once per output file.
If ‘require’, Plotly.js is loaded using require.js. This option assumes that require.js is globally available and that it has been globally configured to know how to find Plotly.js as ‘plotly’. This option is not advised when full_html=True as it will result in a non-functional html file.
If a string that ends in ‘.js’, a script tag is included that references the specified path. This approach can be used to point the resulting HTML file to an alternative CDN or local bundle.
If False, no script tag referencing plotly.js is included. This is useful when the resulting div string will be placed inside an HTML document that already loads plotly.js. This option is not advised when full_html=True as it will result in a non-functional html file.

- **include_mathjax** (*bool* (https://docs.python.org/3/library/functions.html#bool) *or string (default False)*) –
  Specifies how the MathJax.js library is included in the output html div string. MathJax is required in order to display labels with LaTeX typesetting.
  If False, no script tag referencing MathJax.js will be included in the output.
  If ‘cdn’, a script tag that references a MathJax CDN location will be included in the output. HTML div strings generated with this option will be able to display LaTeX typesetting as long as internet access is available.
  If a string that ends in ‘.js’, a script tag is included that references the specified path. This approach can be used to point the resulting HTML div string to an alternative CDN.

- **post_script** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or list* (https://docs.python.org/3/library/stdtypes.html#list) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – JavaScript snippet(s) to be included in the resulting div just after plot creation. The string(s) may include ‘{plot_id}’ placeholders that will then be replaced by the `id` of the div element that the plotly.js figure is associated with. One application for this script is to install custom plotly.js event handlers.

- **full_html** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – If True, produce a string containing a complete HTML document starting with an <html> tag. If False, produce a string containing a single element.

- **animation_opts** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – dict of custom animation parameters to be passed to the function Plotly.animate in Plotly.js. See https://github.com/plotly/plotly.js/blob/master/src/plots/animation_attributes.js (https://github.com/plotly/plotly.js/blob/master/src/plots/animation_attributes.js) for available options. Has no effect if the figure does not contain frames, or auto_play is False.
- **default_width** (*number or str* (https://docs.python.org/3/library/stdtypes.html#str) *(default '100%')*) – The default figure width/height to use if the provided figure does not specify its own layout.width/layout.height property. May be specified in pixels as an integer (e.g. 500), or as a css width style string (e.g. '500px', '100%').
- **default_height** (*number or str* (https://docs.python.org/3/library/stdtypes.html#str) *(default '100%')*) – The default figure width/height to use if the provided figure does not specify its own layout.width/layout.height property. May be specified in pixels as an integer (e.g. 500), or as a css width style string (e.g. '500px', '100%').
- **validate** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if the figure should be validated before being converted to JSON, False otherwise.
- **auto_open** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – If True, open the saved file in a web browser after saving. This argument only applies if `full_html` is True.
- **div_id** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *(default None)*) – If provided, this is the value of the id attribute of the div tag. If None, the id attribute is a UUID.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

`write_image` (**args*, ***kwargs*)

Convert a figure to a static image and write it to a file or writeable object

Parameters:

- **file** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or writeable*) – A string representing a local file path or a writeable object (e.g. a pathlib.Path object or an open file descriptor)
- **format** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The desired image format. One of
  - 'png'
  - 'jpg' or 'jpeg'
  - 'webp'
  - 'svg'
  - 'pdf'
  - 'eps' (Requires the poppler library to be installed)

If not specified and `file` is a string then this will default to the file extension. If not specified and `file` is not a string then this will default to `plotly.io.config.default_format`

- **width** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The width of the exported image in layout pixels. If the `scale` property is 1.0, this will also be the width of the exported image in physical pixels.
  If not specified, will default to `plotly.io.config.default_width`
- **height** (*int* (https://docs.python.org/3/library/functions.html#int) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The height of the exported image in layout pixels. If the `scale` property is 1.0, this will also be the height of the exported image in physical pixels.
  If not specified, will default to `plotly.io.config.default_height`
- **scale** (*int* (https://docs.python.org/3/library/functions.html#int) *or float* (https://docs.python.org/3/library/functions.html#float) *or None* (https://docs.python.org/3/library/constants.html#None)) –
  The scale factor to use when exporting the figure. A scale factor larger than 1.0 will increase the image resolution with respect to the figure's layout pixel dimensions. Whereas as scale factor of less than 1.0 will decrease the image resolution.
  If not specified, will default to `plotly.io.config.default_scale`
- **validate** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True if the figure should be validated before being converted to an image, False otherwise.
- **engine** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) –
  Image export engine to use:
  - "kaleido": Use Kaleido for image export
  - "orca": Use Orca for image export
  - "auto" (default): Use Kaleido if installed, otherwise use orca

Returns:
Return type: None (https://docs.python.org/3/library/constants.html#None)

`write_json` (**args*, ***kwargs*)

Convert a figure to JSON and write it to a file or writeable object

Parameters:

- **file** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or writeable*) – A string representing a local file path or a writeable object (e.g. an open file descriptor)
- **pretty** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default False)*) – True if JSON representation should be pretty-printed, False if representation should be as compact as possible.

- **remove_uids** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if trace UIDs should be omitted from the JSON representation
- **engine** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *(default None)*) –
  The JSON encoding engine to use. One of:
  - ”json” for an encoder based on the built-in Python json module
  - ”orjson” for a fast encoder the requires the orjson package

  If not specified, the default encoder is set to the current value of plotly.io.json.config.default_encoder.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

class plotly.basedatatypes. **BaseFrameHierarchyType** (*plotly_name*, ***kwargs*)

Bases: **plotly.basedatatypes.BasePlotlyType**

Base class for all types in the trace hierarchy

on_change (*callback*, **args*)

Register callback function to be called when certain properties or subproperties of this object are modified.

Callback will be invoked whenever ANY of these properties is modified. Furthermore, the callback will only be invoked once even if multiple properties are modified during the same restyle / relayout / update operation.

Parameters:

- **callback** (*function*) – Function that accepts 1 + len(`args`) parameters. First parameter is this object. Second through last parameters are the property / subpropery values referenced by args.
- **args** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[str|tuple[int|str]]*) – List of property references where each reference may be one of:

  1. A property name string (e.g. ‘foo’) for direct properties
  2. A property path string (e.g. ‘foo[0].bar’) for subproperties
  3. A property path tuple (e.g. (‘foo’, 0, ‘bar’)) for subproperties

- **append** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True if callback should be appended to previously registered callback on the same properties, False if callback should replace previously registered callbacks on the same properties. Defaults to False.

Examples

Register callback that prints out the range extents of the xaxis and yaxis whenever either either of them changes.

```
>>> import plotly.graph_objects as go
>>> fig = go.Figure(go.Scatter(x=[1, 2], y=[1, 0]))
>>> fig.layout.on_change(
...   lambda obj, xrange, yrange: print("%s-%s" % (xrange, yrange)),
...   ('xaxis', 'range'), ('yaxis', 'range'))
```

Returns:
Return type: None (https://docs.python.org/3/library/constants.html#None)

class plotly.basedatatypes. **BaseLayoutHierarchyType** (*plotly_name*, ***kwargs*)

Bases: **plotly.basedatatypes.BasePlotlyType**

Base class for all types in the layout hierarchy

class plotly.basedatatypes. **BaseLayoutType** (*plotly_name*, ***kwargs*)

Bases: **plotly.basedatatypes.BaseLayoutHierarchyType**

Base class for the layout type. The Layout class itself is a code-generated subclass.

class plotly.basedatatypes. **BasePlotlyType** (*plotly_name*, ***kwargs*)

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

BasePlotlyType is the base class for all objects in the trace, layout, and frame object hierarchies

property **figure**

Reference to the top-level Figure or FigureWidget that this object belongs to. None if the object does not belong to a Figure

Returns:

Return type:	Union[BaseFigure (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure), None (https://docs.python.org/3/library/constants.html#None)]

on_change (*callback*, **args*, ***kwargs*)

Register callback function to be called when certain properties or subproperties of this object are modified.

Callback will be invoked whenever ANY of these properties is modified. Furthermore, the callback will only be invoked once even if multiple properties are modified during the same restyle / relayout / update operation.

Parameters:

- **callback** (*function*) – Function that accepts 1 + len(`args`) parameters. First parameter is this object. Second through last parameters are the property / subpropery values referenced by args.
- **args** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[str|tuple[int|str]]*) – List of property references where each reference may be one of:

1. A property name string (e.g. ‘foo’) for direct properties
2. A property path string (e.g. ‘foo[0].bar’) for subproperties
3. A property path tuple (e.g. (‘foo’, 0, ‘bar’)) for subproperties

- **append** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True if callback should be appended to previously registered callback on the same properties, False if callback should replace previously registered callbacks on the same properties. Defaults to False.

Examples

Register callback that prints out the range extents of the xaxis and yaxis whenever either either of them changes.

```
>>> import plotly.graph_objects as go
>>> fig = go.Figure(go.Scatter(x=[1, 2], y=[1, 0]))
>>> fig.layout.on_change(
...   lambda obj, xrange, yrange: print("%s-%s" % (xrange, yrange)),
...   ('xaxis', 'range'), ('yaxis', 'range'))
```

Returns:
Return type: None (https://docs.python.org/3/library/constants.html#None)

property **parent**

Return the object’s parent, or None if the object has no parent :returns: :rtype: BasePlotlyType|BaseFigure

property **plotly_name**

The plotly name of the object

Returns:
Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

pop (*key*, **args*)

Remove the value associated with the specified key and return it

Parameters:
- **key** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Property name
- **dflt** – The default value to return if key was not found in object

Returns: The removed value that was previously associated with key
Return type: value
Raises: **KeyError** (https://docs.python.org/3/library/exceptions.html#KeyError) – If key is not in object and no dflt argument specified

to_json (**args*, ***kwargs*)

Convert object to a JSON string representation

Parameters:

- **validate** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if the object should be validated before being converted to JSON, False otherwise.
- **pretty** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default False)*) – True if JSON representation should be pretty-printed, False if representation should be as compact as possible.
- **remove_uids** (*bool* (https://docs.python.org/3/library/functions.html#bool) *(default True)*) – True if trace UIDs should be omitted from the JSON representation
- **engine** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *(default None)*) –

  The JSON encoding engine to use. One of:

  - ”json” for an encoder based on the built-in Python json module
  - ”orjson” for a fast encoder the requires the orjson package

  If not specified, the default encoder is set to the current value of plotly.io.json.config.default_encoder.

Returns: Representation of object as a JSON string
Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

`to_plotly_json` ()

Return plotly JSON representation of object as a Python dict

Note: May include some JSON-invalid data types, use the `PlotlyJSONEncoder` util or the `to_json` method to encode to a string.

Returns:
Return type: dict (https://docs.python.org/3/library/stdtypes.html#dict)

`update` (*dict1=None*, *overwrite=False*, ***kwargs*)

Update the properties of an object with a dict and/or with keyword arguments.

This recursively updates the structure of the original object with the values in the input dict / keyword arguments.

Parameters:

- **dict1** (*dict* (https://docs.python.org/3/library/stdtypes.html#dict)) – Dictionary of properties to be updated
- **overwrite** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If True, overwrite existing properties. If False, apply updates to existing properties recursively, preserving existing properties that are not specified in the update operation.
- **kwargs** – Keyword/value pair of properties to be updated

Returns: Updated plotly object
Return type: BasePlotlyType

class `plotly.basedatatypes.` **`BaseTraceHierarchyType`** (*plotly_name*, ***kwargs*)

Bases: **plotly.basedatatypes.BasePlotlyType**

Base class for all types in the trace hierarchy

class plotly.basedatatypes. **BaseTraceType** (*plotly_name*, ***kwargs*)

Bases: **plotly.basedatatypes.BaseTraceHierarchyType**

Base class for the all trace types.

Specific trace type classes (Scatter, Bar, etc.) are code generated as subclasses of this class.

on_click (*callback*, *append=False*)

Register function to be called when the user clicks on one or more points in this trace.

Note: Callbacks will only be triggered when the trace belongs to a instance of plotly.graph_objects.FigureWidget and it is displayed in an ipywidget context. Callbacks will not be triggered on figures that are displayed using plot/iplot.

Parameters:

- **callback** –
  Callable function that accepts 3 arguments
  - this trace
  - plotly.callbacks.Points object
  - plotly.callbacks.InputDeviceState object
- **append** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If False (the default), this callback replaces any previously defined on_click callbacks for this trace. If True, this callback is appended to the list of any previously defined callbacks.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

Examples

```
>>> import plotly.graph_objects as go
>>> from plotly.callbacks import Points, InputDeviceState
>>> points, state = Points(), InputDeviceState()
```

```
>>> def click_fn(trace, points, state):
...     inds = points.point_inds
...     # Do something
```

```
>>> trace = go.Scatter(x=[1, 2], y=[3, 0])
>>> trace.on_click(click_fn)
```

Note: The creation of the `points` and `state` objects is optional, it's simply a convenience to help the text editor perform completion on the arguments inside `click_fn`

on_deselect (*callback*, *append=False*)

Register function to be called when the user deselects points in this trace using doubleclick.

Note: Callbacks will only be triggered when the trace belongs to a instance of plotly.graph_objects.FigureWidget and it is displayed in an ipywidget context. Callbacks will not be triggered on figures that are displayed using plot/iplot.

Parameters:

- **callback** –
  Callable function that accepts 3 arguments
  - this trace
  - plotly.callbacks.Points object
- **append** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If False (the default), this callback replaces any previously defined on_deselect callbacks for this trace. If True, this callback is appended to the list of any previously defined callbacks.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

Examples

```
>>> import plotly.graph_objects as go
>>> from plotly.callbacks import Points
>>> points = Points()
```

```
>>> def deselect_fn(trace, points):
...     inds = points.point_inds
...     # Do something
```

```
>>> trace = go.Scatter(x=[1, 2], y=[3, 0])
>>> trace.on_deselect(deselect_fn)
```

Note: The creation of the `points` object is optional, it's simply a convenience to help the text editor perform completion on the `points` arguments inside `selection_fn`

`on_hover` (*callback*, *append=False*)

Register function to be called when the user hovers over one or more points in this trace

Note: Callbacks will only be triggered when the trace belongs to a instance of plotly.graph_objects.FigureWidget and it is displayed in an ipywidget context. Callbacks will not be triggered on figures that are displayed using plot/iplot.

Parameters:

- **callback** –
  Callable function that accepts 3 arguments
  - this trace
  - plotly.callbacks.Points object
  - plotly.callbacks.InputDeviceState object

- **append** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If False (the default), this callback replaces any previously defined on_hover callbacks for this trace. If True, this callback is appended to the list of any previously defined callbacks.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

Examples

```
>>> import plotly.graph_objects as go
>>> from plotly.callbacks import Points, InputDeviceState
>>> points, state = Points(), InputDeviceState()
```

```
>>> def hover_fn(trace, points, state):
...     inds = points.point_inds
...     # Do something
```

```
>>> trace = go.Scatter(x=[1, 2], y=[3, 0])
>>> trace.on_hover(hover_fn)
```

Note: The creation of the `points` and `state` objects is optional, it's simply a convenience to help the text editor perform completion on the arguments inside `hover_fn`

`on_selection` (*callback*, *append=False*)

Register function to be called when the user selects one or more points in this trace.

Note: Callbacks will only be triggered when the trace belongs to a instance of plotly.graph_objects.FigureWidget and it is displayed in an ipywidget context. Callbacks will not be triggered on figures that are displayed using plot/iplot.

Parameters:

- **callback** –
  Callable function that accepts 4 arguments
  - this trace
  - plotly.callbacks.Points object
  - plotly.callbacks.BoxSelector or plotly.callbacks.LassoSelector
- **append** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If False (the default), this callback replaces any previously defined on_selection callbacks for this trace. If True, this callback is appended to the list of any previously defined callbacks.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

Examples

```
>>> import plotly.graph_objects as go
>>> from plotly.callbacks import Points
>>> points = Points()
```

```
>>> def selection_fn(trace, points, selector):
...     inds = points.point_inds
...     # Do something
```

```
>>> trace = go.Scatter(x=[1, 2], y=[3, 0])
>>> trace.on_selection(selection_fn)
```

Note: The creation of the `points` object is optional, it's simply a convenience to help the text editor perform completion on the `points` arguments inside `selection_fn`

`on_unhover` (*callback*, *append=False*)

Register function to be called when the user unhovers away from one or more points in this trace.

Note: Callbacks will only be triggered when the trace belongs to a instance of plotly.graph_objects.FigureWidget and it is displayed in an ipywidget context. Callbacks will not be triggered on figures that are displayed using plot/iplot.

Parameters:

- **callback** –
  Callable function that accepts 3 arguments
  - this trace
  - plotly.callbacks.Points object
  - plotly.callbacks.InputDeviceState object
- **append** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – If False (the default), this callback replaces any previously defined on_unhover callbacks for this trace. If True, this callback is appended to the list of any previously defined callbacks.

Returns:

Return type: None (https://docs.python.org/3/library/constants.html#None)

Examples

```
>>> import plotly.graph_objects as go
>>> from plotly.callbacks import Points, InputDeviceState
>>> points, state = Points(), InputDeviceState()
```

```
>>> def unhover_fn(trace, points, state):
...     inds = points.point_inds
...     # Do something
```

```
>>> trace = go.Scatter(x=[1, 2], y=[3, 0])
>>> trace.on_unhover(unhover_fn)
```

Note: The creation of the `points` and `state` objects is optional, it's simply a convenience to help the text editor perform completion on the arguments inside `unhover_fn`

property **uid**

plotly.basewidget module

class `plotly.basewidget.` **BaseFigureWidget** (***kwargs: Any*)

Bases: **plotly.basedatatypes.BaseFigure** (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure), **ipywidgets.widgets.domwidget.DOMWidget**

Base class for FigureWidget. The FigureWidget class is code-generated as a subclass

property **frames**

The `frames` property is a tuple of the figure's frame objects

Returns:

Return type: tuple (https://docs.python.org/3/library/stdtypes.html#tuple) [plotly.graph_objects.Frame (plotly.graph_objects.html#plotly.graph_objects.Frame)]

on_edits_completed (*fn*)

Register a function to be called after all pending trace and layout edit operations have completed

If there are no pending edit operations then function is called immediately

Parameters: **fn** (*callable*) – Function of zero arguments to be called when all pending edit operations have completed

plotly.callbacks module

class `plotly.callbacks.` **BoxSelector** (*xrange=None*, *yrange=None*, ***_*)

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

property **type**

The selector's type

Returns:

Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

property **xrange**

x-axis range extents of the box selection

Returns:

Return type: (float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float))

property **yrange**

y-axis range extents of the box selection

Returns:
Return type: (float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float))

class plotly.callbacks. **InputDeviceState** (*ctrl=None*, *alt=None*, *shift=None*, *meta=None*, *button=None*, *buttons=None*, ***_*)

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

property **alt**

Whether alt key pressed

Returns:
Return type: bool (https://docs.python.org/3/library/functions.html#bool)

property **button**

Integer code for the button that was pressed on the mouse to trigger the event

- 0: Main button pressed, usually the left button or the un-initialized state
- 1: Auxiliary button pressed, usually the wheel button or the middle button (if present)
- 2: Secondary button pressed, usually the right button
- 3: Fourth button, typically the Browser Back button
- 4: Fifth button, typically the Browser Forward button

Returns:
Return type: int (https://docs.python.org/3/library/functions.html#int)

property **buttons**

Integer code for which combination of buttons are pressed on the mouse when the event is triggered.

- 0: No button or un-initialized
- 1: Primary button (usually left)
- 2: Secondary button (usually right)
- 4: Auxilary button (usually middle or mouse wheel button)
- 8: 4th button (typically the “Browser Back” button)
- 16: 5th button (typically the “Browser Forward” button)

Combinations of buttons are represented as the decimal form of the bitmask of the values above.

For example, pressing both the primary (1) and auxilary (4) buttons will result in a code of 5

Returns:
Return type: int (https://docs.python.org/3/library/functions.html#int)

property **ctrl**

Whether ctrl key pressed

Returns:
Return type: bool (https://docs.python.org/3/library/functions.html#bool)

property **meta**

Whether meta key pressed

Returns:
Return type: bool (https://docs.python.org/3/library/functions.html#bool)

property **shift**

Whether shift key pressed

Returns:
Return type: bool (https://docs.python.org/3/library/functions.html#bool)

class plotly.callbacks. **LassoSelector** (*xs=None*, *ys=None*, ***_*)

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

property **type**

The selector's type

Returns:
Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

property **xs**

list of x-axis coordinates of each point in the lasso selection boundary

Returns:
Return type: list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

property **ys**

list of y-axis coordinates of each point in the lasso selection boundary

Returns:
Return type: list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

class plotly.callbacks. **Points** (*point_inds=[]*, *xs=[]*, *ys=[]*, *trace_name=None*, *trace_index=None*)

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

property **point_inds**

List of selected indexes into the trace's points

Returns:
Return type: list (https://docs.python.org/3/library/stdtypes.html#list)[int (https://docs.python.org/3/library/functions.html#int)]

property **trace_index**

Index of the trace in the figure

Returns:
Return type: int (https://docs.python.org/3/library/functions.html#int)

property **trace_name**

Name of the trace

Returns:
Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

property **xs**

List of x-coordinates of selected points

Returns:
Return type: list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

property **ys**

List of y-coordinates of selected points

Returns:
Return type: list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

plotly.config module

plotly.conftest module

plotly.conftest. **pytest_ignore_collect** (*path*)

plotly.dashboard_objs module

plotly.exceptions module

plotly.files module

plotly.grid_objs module

plotly.missing_ipywidgets module

class plotly.missing_ipywidgets. **FigureWidget** (**args*, ***kwargs*)

Bases: **plotly.basedatatypes.BaseFigure** (plotly.basedatatypes.BaseFigure.html#plotly.basedatatypes.BaseFigure)

FigureWidget stand-in for use when ipywidgets is not installed. The only purpose of this class is to provide something to import as plotly.graph_objects.FigureWidget when ipywidgets is not installed. This class simply raises an informative error message when the constructor is called

plotly.optional_imports module

plotly.presentation_objs module

plotly.serializers module

plotly.session module

plotly.shapeannotation module

plotly.shapeannotation. **annotation_params_for_line** (*shape_type*, *shape_args*, *position*)

plotly.shapeannotation. **annotation_params_for_rect** (*shape_type*, *shape_args*, *position*)

plotly.shapeannotation. **axis_spanning_shape_annotation** (*annotation*, *shape_type*, *shape_args*, *kwargs*)

annotation: a go.layout.Annotation object, a dict describing an annotation, or None shape_type: one of ‘vline’, ‘hline’, ‘vrect’, ‘hrect’ and determines how the

> x, y, xanchor, and yanchor values are set.

shape_args: the parameters used to draw the shape, which are used to place the annotation kwargs: a dictionary that was the kwargs of a

> _process_multiple_axis_spanning_shapes spanning shapes call. Items in this dict whose keys start with ‘annotation_’ will be extracted and the keys with the ‘annotation_’ part stripped off will be used to assign properties of the new annotation.

Property precedence: The annotation’s x, y, xanchor, and yanchor properties are set based on the shape_type argument. Each property already specified in the annotation or through kwargs will be left as is (not replaced by the value computed using shape_type). Note that the xref and yref

properties will in general get overwritten if the result of this function is passed to an add_annotation called with the row and col parameters specified.

Returns an annotation populated with fields based on the annotation_position, annotation_ prefixed kwargs or the original annotation passed in to this function.

plotly.shapeannotation. **split_dict_by_key_prefix** (*d*, *prefix*)

Returns two dictionaries, one containing all the items whose keys do not start with a prefix and another containing all the items whose keys do start with the prefix. Note that the prefix is not removed from the keys.

plotly.subplots module

plotly.subplots. **make_subplots** (*rows=1*, *cols=1*, *shared_xaxes=False*, *shared_yaxes=False*, *start_cell='top-left'*, *print_grid=False*, *horizontal_spacing=None*, *vertical_spacing=None*, *subplot_titles=None*, *column_widths=None*, *row_heights=None*, *specs=None*, *insets=None*, *column_titles=None*, *row_titles=None*, *x_title=None*, *y_title=None*, *figure=None*, ***kwargs*) → plotly.graph_objects._figure.Figure

Return an instance of plotly.graph_objects.Figure with predefined subplots configured in 'layout'.

Parameters:

- **rows** (*int* (https://docs.python.org/3/library/functions.html#int) *(default 1)*) – Number of rows in the subplot grid. Must be greater than zero.
- **cols** (*int* (https://docs.python.org/3/library/functions.html#int) *(default 1)*) – Number of columns in the subplot grid. Must be greater than zero.
- **shared_xaxes** (*boolean or str* (https://docs.python.org/3/library/stdtypes.html#str) *(default False)*) –
  Assign shared (linked) x-axes for 2D cartesian subplots
  - True or 'columns': Share axes among subplots in the same column
  - 'rows': Share axes among subplots in the same row
  - 'all': Share axes across all subplots in the grid.
- **shared_yaxes** (*boolean or str* (https://docs.python.org/3/library/stdtypes.html#str) *(default False)*) –
  Assign shared (linked) y-axes for 2D cartesian subplots
  - 'columns': Share axes among subplots in the same column
  - True or 'rows': Share axes among subplots in the same row
  - 'all': Share axes across all subplots in the grid.
- **start_cell** (*'bottom-left' or 'top-left' (default 'top-left')*) –
  Choose the starting cell in the subplot grid used to set the domains_grid of the subplots.

- 'top-left': Subplots are numbered with (1, 1) in the top left corner
- 'bottom-left': Subplots are numbererd with (1, 1) in the bottom left corner

- **print_grid** (*boolean (default True):*) – If True, prints a string representation of the plot grid. Grid may also be printed using the `Figure.print_grid()` method on the resulting figure.
- **horizontal_spacing** (*float* (https://docs.python.org/3/library/functions.html#float) *(default 0.2 / cols)*) –
  Space between subplot columns in normalized plot coordinates. Must be a float between 0 and 1.
  Applies to all columns (use 'specs' subplot-dependents spacing)
- **vertical_spacing** (*float* (https://docs.python.org/3/library/functions.html#float) *(default 0.3 / rows)*) –
  Space between subplot rows in normalized plot coordinates. Must be a float between 0 and 1.
  Applies to all rows (use 'specs' subplot-dependents spacing)
- **subplot_titles** (*list of str or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) –
  Title of each subplot as a list in row-major ordering.
  Empty strings ("") can be included in the list if no subplot title is desired in that space so that the titles are properly indexed.
- **specs** (*list of lists of dict or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) –
  Per subplot specifications of subplot type, row/column spanning, and spacing.
  ex1: specs=[[{}, {}], [{'colspan': 2}, None]]
  ex2: specs=[[{'rowspan': 2}, {}], [None, {}]]
  - Indices of the outer list correspond to subplot grid rows starting from the top, if start_cell='top-left', or bottom, if start_cell='bottom-left'. The number of rows in 'specs' must be equal to 'rows'.
  - Indices of the inner lists correspond to subplot grid columns starting from the left. The number of columns in 'specs' must be equal to 'cols'.
  - Each item in the 'specs' list corresponds to one subplot in a subplot grid. (N.B. The subplot grid has exactly 'rows' times 'cols' cells.)
  - Use None for a blank a subplot cell (or to move past a col/row span).
  - Note that specs[0][0] has the specs of the 'start_cell' subplot.
  - Each item in 'specs' is a dictionary.
    The available keys are: * type (string, default 'xy'): Subplot type. One of

- 'xy': 2D Cartesian subplot type for scatter, bar, etc.
- 'scene': 3D Cartesian subplot for scatter3d, cone, etc.
- 'polar': Polar subplot for scatterpolar, barpolar, etc.
- 'ternary': Ternary subplot for scatterternary
- 'map': Map subplot for scattermap
- 'mapbox': Mapbox subplot for scattermapbox
- 'domain': Subplot type for traces that are individually positioned. pie, parcoords, parcats, etc.
- trace type: A trace type which will be used to determine the appropriate subplot type for that trace

- secondary_y (bool, default False): If True, create a secondary y-axis positioned on the right side of the subplot. Only valid if type='xy'.
- colspan (int, default 1): number of subplot columns for this subplot to span.
- rowspan (int, default 1): number of subplot rows for this subplot to span.
- l (float, default 0.0): padding left of cell
- r (float, default 0.0): padding right of cell
- t (float, default 0.0): padding right of cell
- b (float, default 0.0): padding bottom of cell

  - Note: Use 'horizontal_spacing' and 'vertical_spacing' to adjust the spacing in between the subplots.

- **insets** (*list of dict or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*:) –
  Inset specifications. Insets are subplots that overlay grid subplots
  - Each item in 'insets' is a dictionary.
    The available keys are:
    - cell (tuple, default=(1,1)): (row, col) index of the subplot cell to overlay inset axes onto.
    - type (string, default 'xy'): Subplot type
    - l (float, default=0.0): padding left of inset in fraction of cell width
    - w (float or 'to_end', default='to_end') inset width

in fraction of cell width (‘to_end’: to cell right edge)

- b (float, default=0.0): padding bottom of inset
  in fraction of cell height

- h (float or ‘to_end’, default=’to_end’) inset height
  in fraction of cell height (‘to_end’: to cell top edge)

- **column_widths** (*list of numbers or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – list of length `cols` of the relative widths of each column of subplots. Values are normalized internally and used to distribute overall width of the figure (excluding padding) among the columns.
  For backward compatibility, may also be specified using the `column_width` keyword argument.
- **row_heights** (*list of numbers or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – list of length `rows` of the relative heights of each row of subplots. If start_cell=’top-left’ then row heights are applied top to bottom. Otherwise, if start_cell=’bottom-left’ then row heights are applied bottom to top.
  For backward compatibility, may also be specified using the `row_width` kwarg. If specified as `row_width`, then the width values are applied from bottom to top regardless of the value of start_cell. This matches the legacy behavior of the `row_width` argument.
- **column_titles** (*list of str or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – list of length `cols` of titles to place above the top subplot in each column.
- **row_titles** (*list of str or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – list of length `rows` of titles to place on the right side of each row of subplots. If start_cell=’top-left’ then row titles are applied top to bottom. Otherwise, if start_cell=’bottom-left’ then row titles are applied bottom to top.
- **x_title** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Title to place below the bottom row of subplots, centered horizontally
- **y_title** (*str* (https://docs.python.org/3/library/stdtypes.html#str) *or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – Title to place to the left of the left column of subplots, centered vertically
- **figure** (*go.Figure or None* (https://docs.python.org/3/library/constants.html#None) *(default None)*) – If None, a new go.Figure instance will be created and its axes will be populated with those corresponding to the requested subplot geometry and this new figure will be returned. If a go.Figure instance, the axes will be added to the layout of this figure and this figure will be returned. If the figure already contains axes, they will be overwritten.

Examples

Example 1:

```
>>> # Stack two subplots vertically, and add a scatter trace to each
>>> from plotly.subplots import make_subplots
>>> import plotly.graph_objects as go
>>> fig = make_subplots(rows=2)
```

This is the format of your plot grid: [(1,1) xaxis1,yaxis1] [(2,1) xaxis2,yaxis2]

```
>>> fig.add_scatter(y=[2, 1, 3], row=1, col=1)
Figure(...)
>>> fig.add_scatter(y=[1, 3, 2], row=2, col=1)
Figure(...)
```

or see Figure.append_trace

Example 2:

```
>>> # Stack a scatter plot
>>> fig = make_subplots(rows=2, shared_xaxes=True)
```

This is the format of your plot grid: [(1,1) xaxis1,yaxis1] [(2,1) xaxis2,yaxis2]

```
>>> fig.add_scatter(y=[2, 1, 3], row=1, col=1)
Figure(...)
>>> fig.add_scatter(y=[1, 3, 2], row=2, col=1)
Figure(...)
```

Example 3:

```
>>> # irregular subplot layout (more examples below under 'specs')
>>> fig = make_subplots(rows=2, cols=2,
...                     specs=[[{}, {}],
...                     [{'colspan': 2}, None]])
```

This is the format of your plot grid: [(1,1) xaxis1,yaxis1] [(1,2) xaxis2,yaxis2] [(2,1) xaxis3,yaxis3 -]

```
>>> fig.add_trace(go.Scatter(x=[1,2,3], y=[2,1,2]), row=1, col=1)
Figure(...)
>>> fig.add_trace(go.Scatter(x=[1,2,3], y=[2,1,2]), row=1, col=2)
Figure(...)
>>> fig.add_trace(go.Scatter(x=[1,2,3], y=[2,1,2]), row=2, col=1)
Figure(...)
```

Example 4:

```
>>> # insets
>>> fig = make_subplots(insets=[{'cell': (1,1), 'l': 0.7, 'b': 0.3}])
```

This is the format of your plot grid: [(1,1) xaxis1,yaxis1]

With insets: [xaxis2,yaxis2] over [(1,1) xaxis1,yaxis1]

```
>>> fig.add_scatter(x=[1,2,3], y=[2,1,1])
Figure(...)
>>> fig.add_scatter(x=[1,2,3], y=[2,1,2], xaxis='x2', yaxis='y2')
Figure(...)
```

Example 5:

```
>>> # include subplot titles
>>> fig = make_subplots(rows=2, subplot_titles=('Plot 1','Plot 2'))
```

This is the format of your plot grid: [(1,1) x1,y1] [(2,1) x2,y2]

```
>>> fig.add_scatter(x=[1,2,3], y=[2,1,2], row=1, col=1)
Figure(...)
>>> fig.add_bar(x=[1,2,3], y=[2,1,2], row=2, col=1)
Figure(...)
```

Example 6:

Subplot with mixed subplot types

```
>>> fig = make_subplots(rows=2, cols=2,
...                     specs=[[{'type': 'xy'},    {'type': 'polar'}],
...                            [{'type': 'scene'}, {'type': 'ternary'}]])
```

```
>>> fig.add_traces(
...     [go.Scatter(y=[2, 3, 1]),
...      go.Scatterpolar(r=[1, 3, 2], theta=[0, 45, 90]),
...      go.Scatter3d(x=[1, 2, 1], y=[2, 3, 1], z=[0, 3, 5]),
...      go.Scatterternary(a=[0.1, 0.2, 0.1],
...                        b=[0.2, 0.3, 0.1],
...                        c=[0.7, 0.5, 0.8])],
...     rows=[1, 1, 2, 2],
...     cols=[1, 2, 1, 2])
Figure(...)
```

plotly.tools module

tools

Functions that USERS will possibly want access to.

class plotly.tools. **FigureFactory**

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

static **create_2D_density** (**args*, ***kwargs*)

static **create_annotated_heatmap** (**args*, ***kwargs*)

static **create_candlestick** (**args*, ***kwargs*)

static **create_dendrogram** (**args*, ***kwargs*)

static **create_distplot** (**args*, ***kwargs*)

static **create_facet_grid** (**args*, ***kwargs*)

static **create_gantt** (**args*, ***kwargs*)

static **create_ohlc** (**args*, ***kwargs*)

static **create_quiver** (**args*, ***kwargs*)

static **create_scatterplotmatrix** (**args*, ***kwargs*)

static **create_streamline** (**args*, ***kwargs*)

static **create_table** (**args*, ***kwargs*)

static **create_trisurf** (**args*, ***kwargs*)

static **create_violin** (**args*, ***kwargs*)

plotly.tools. **get_config_plotly_server_url** ()

Function to get the .config file's 'plotly_domain' without importing the chart_studio package. This property is needed to compute the default value of the plotly.js config plotlyServerURL, so it is independent of the chart_studio integration and still needs to live in

Returns:

Return type: str (https://docs.python.org/3/library/stdtypes.html#str)

plotly.tools. **get_graph_obj** (*obj*, *obj_type=None*)

Returns a new graph object.

OLD FUNCTION: this will *silently* strip out invalid pieces of the object. NEW FUNCTION: no striping of invalid pieces anymore - only raises error

> on unrecognized graph_objects

plotly.tools. **get_subplots** (*rows=1*, *columns=1*, *print_grid=False*, ***kwargs*)

Return a dictionary instance with the subplots set in 'layout'.

Example 1: # stack two subplots vertically fig = tools.get_subplots(rows=2) fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x1', yaxis='y1')] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x2', yaxis='y2')]

Example 2: # print out string showing the subplot grid you've put in the layout fig = tools.get_subplots(rows=3, columns=2, print_grid=True)

Keywords arguments with constant defaults:

rows (kwarg, int greater than 0, default=1):
Number of rows, evenly spaced vertically on the figure.

columns (kwarg, int greater than 0, default=1):
Number of columns, evenly spaced horizontally on the figure.

horizontal_spacing (kwarg, float in [0,1], default=0.1):
Space between subplot columns. Applied to all columns.

vertical_spacing (kwarg, float in [0,1], default=0.05):
Space between subplot rows. Applied to all rows.

print_grid (kwarg, True | False, default=False):
If True, prints a tab-delimited string representation of your plot grid.

Keyword arguments with variable defaults:

horizontal_spacing (kwarg, float in [0,1], default=0.2 / columns):
Space between subplot columns.

vertical_spacing (kwarg, float in [0,1], default=0.3 / rows):
Space between subplot rows.

plotly.tools. **make_subplots** (*rows=1*, *cols=1*, *shared_xaxes=False*, *shared_yaxes=False*, *start_cell='top-left'*, *print_grid=None*, ***kwargs*)

Return an instance of plotly.graph_objects.Figure with the subplots domain set in 'layout'.

Example 1: # stack two subplots vertically fig = tools.make_subplots(rows=2)

This is the format of your plot grid: [(1,1) x1,y1] [(2,1) x2,y2]

fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2])] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x2', yaxis='y2')]

or see Figure.append_trace

Example 2: # subplots with shared x axes fig = tools.make_subplots(rows=2, shared_xaxes=True)

This is the format of your plot grid: [(1,1) x1,y1] [(2,1) x1,y2]

fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2])] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], yaxis='y2')]

Example 3: # irregular subplot layout (more examples below under 'specs') fig = tools.make_subplots(rows=2, cols=2,

```
    specs=[[{}, {}],
        [{'colspan': 2}, None]])
```

This is the format of your plot grid! [(1,1) x1,y1] [(1,2) x2,y2] [(2,1) x3,y3 -]

fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2])] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x2', yaxis='y2')] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x3', yaxis='y3')]

Example 4: # insets fig = tools.make_subplots(insets=[{'cell': (1,1), 'l': 0.7, 'b': 0.3}])

This is the format of your plot grid! [(1,1) x1,y1]

With insets: [x2,y2] over [(1,1) x1,y1]

fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2])] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x2', yaxis='y2')]

Example 5: # include subplot titles fig = tools.make_subplots(rows=2, subplot_titles=('Plot 1','Plot 2'))

This is the format of your plot grid: [(1,1) x1,y1] [(2,1) x2,y2]

fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2])] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x2', yaxis='y2')]

Example 6: # Include subplot title on one plot (but not all) fig = tools.make_subplots(insets=[{'cell': (1,1), 'l': 0.7, 'b': 0.3}],

```
    subplot_titles=('','Inset'))
```

This is the format of your plot grid! [(1,1) x1,y1]

With insets: [x2,y2] over [(1,1) x1,y1]

fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2])] fig['data'] += [Scatter(x=[1,2,3], y=[2,1,2], xaxis='x2', yaxis='y2')]

Keywords arguments with constant defaults:

rows (kwarg, int greater than 0, default=1):
    Number of rows in the subplot grid.

cols (kwarg, int greater than 0, default=1):
    Number of columns in the subplot grid.

shared_xaxes (kwarg, boolean or list, default=False)

Assign shared x axes. If True, subplots in the same grid column have one common shared x-axis at the bottom of the gird.

To assign shared x axes per subplot grid cell (see 'specs'), send list (or list of lists, one list per shared x axis) of cell index tuples.

shared_yaxes (kwarg, boolean or list, default=False)

Assign shared y axes. If True, subplots in the same grid row have one common shared y-axis on the left-hand side of the gird.

To assign shared y axes per subplot grid cell (see 'specs'), send list (or list of lists, one list per shared y axis) of cell index tuples.

start_cell (kwarg, 'bottom-left' or 'top-left', default='top-left')

Choose the starting cell in the subplot grid used to set the domains of the subplots.

print_grid (kwarg, boolean, default=True):

If True, prints a tab-delimited string representation of your plot grid.

Keyword arguments with variable defaults:

horizontal_spacing (kwarg, float in [0,1], default=0.2 / cols):

Space between subplot columns. Applies to all columns (use 'specs' subplot-dependents spacing)

vertical_spacing (kwarg, float in [0,1], default=0.3 / rows):

Space between subplot rows. Applies to all rows (use 'specs' subplot-dependents spacing)

subplot_titles (kwarg, list of strings, default=empty list):

Title of each subplot. "" can be included in the list if no subplot title is desired in that space so that the titles are properly indexed.

specs (kwarg, list of lists of dictionaries):

Subplot specifications.

ex1: specs=[[{}, {}], [{'colspan': 2}, None]]

ex2: specs=[[{'rowspan': 2}, {}], [None, {}]]

- Indices of the outer list correspond to subplot grid rows starting from the bottom. The number of rows in 'specs' must be equal to 'rows'.
- Indices of the inner lists correspond to subplot grid columns starting from the left. The number of columns in 'specs' must be equal to 'cols'.
- Each item in the 'specs' list corresponds to one subplot in a subplot grid. (N.B. The subplot grid has exactly 'rows' times 'cols' cells.)
- Use None for blank a subplot cell (or to move pass a col/row span).
- Note that specs[0][0] has the specs of the 'start_cell' subplot.
- Each item in 'specs' is a dictionary.

  The available keys are:

  - is_3d (boolean, default=False): flag for 3d scenes
  - colspan (int, default=1): number of subplot columns

    for this subplot to span.

- rowspan (int, default=1): number of subplot rows
  for this subplot to span.

- l (float, default=0.0): padding left of cell
- r (float, default=0.0): padding right of cell
- t (float, default=0.0): padding right of cell
- b (float, default=0.0): padding bottom of cell

- Use ‘horizontal_spacing’ and ‘vertical_spacing’ to adjust the spacing in between the subplots.

insets (kwarg, list of dictionaries):

Inset specifications.

- Each item in ‘insets’ is a dictionary.
  The available keys are:
  - cell (tuple, default=(1,1)): (row, col) index of the
    subplot cell to overlay inset axes onto.

  - is_3d (boolean, default=False): flag for 3d scenes
  - l (float, default=0.0): padding left of inset
    in fraction of cell width

  - w (float or ‘to_end’, default=’to_end’) inset width
    in fraction of cell width (‘to_end’: to cell right edge)

  - b (float, default=0.0): padding bottom of inset
    in fraction of cell height

  - h (float or ‘to_end’, default=’to_end’) inset height
    in fraction of cell height (‘to_end’: to cell top edge)

column_width (kwarg, list of numbers)

Column_width specifications

- Functions similarly to `column_width` of `plotly.graph_objects.Table`. Specify a list that contains numbers where the amount of numbers in the list is equal to `cols`.
- The numbers in the list indicate the proportions that each column domains take across the full horizontal domain excluding padding.
- For example, if columns_width=[3, 1], horizontal_spacing=0, and cols=2, the domains for each column would be [0. 0.75] and [0.75, 1]

row_width (kwargs, list of numbers)

Row_width specifications

- Functions similarly to `column_width`. Specify a list that contains numbers where the amount of numbers in the list is equal to `rows`.

- The numbers in the list indicate the proportions that each row domains take along the full vertical domain excluding padding.
- For example, if row_width=[3, 1], vertical_spacing=0, and cols=2, the domains for each row from top to botton would be [0. 0.75] and [0.75, 1]

plotly.tools. **mpl_to_plotly** (*fig*, *resize=False*, *strip_style=False*, *verbose=False*)

Convert a matplotlib figure to plotly dictionary and send.

All available information about matplotlib visualizations are stored within a matplotlib.figure.Figure object. You can create a plot in python using matplotlib, store the figure object, and then pass this object to the fig_to_plotly function. In the background, mplexporter is used to crawl through the mpl figure object for appropriate information. This information is then systematically sent to the PlotlyRenderer which creates the JSON structure used to make plotly visualizations. Finally, these dictionaries are sent to plotly and your browser should open up a new tab for viewing! Optionally, if you're working in IPython, you can set notebook=True and the PlotlyRenderer will call plotly.iplot instead of plotly.plot to have the graph appear directly in the IPython notebook.

Note, this function gives the user access to a simple, one-line way to render an mpl figure in plotly. If you need to trouble shoot, you can do this step manually by NOT running this fuction and entereing the following:

from plotly.matplotlylib import mplexporter, PlotlyRenderer

create an mpl figure and store it under a varialble 'fig'

renderer = PlotlyRenderer() exporter = mplexporter.Exporter(renderer) exporter.run(fig) ==========

You can then inspect the JSON structures by accessing these:

renderer.layout – a plotly layout dictionary renderer.data – a list of plotly data dictionaries

plotly.tools. **return_figure_from_figure_or_data** (*figure_or_data*, *validate_figure*)

plotly.tools. **warning_on_one_line** (*message*, *category*, *filename*, *lineno*, *file=None*, *line=None*)

plotly.utils module

class plotly.utils. **ElidedPrettyPrinter** (**args*, ***kwargs*)

Bases: **pprint.PrettyPrinter** (https://docs.python.org/3/library/pprint.html#pprint.PrettyPrinter)

PrettyPrinter subclass that elides long lists/arrays/strings

class plotly.utils. **ElidedWrapper** (*v*, *threshold*, *indent*)

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

Helper class that wraps values of certain types and produces a custom __repr__() that may be elided and is suitable for use during pretty printing

static **is_wrappable**(*v*)

plotly.utils. **decode_unicode**(*coll*)

plotly.utils. **get_by_path**(*obj*, *path*)

Iteratively get on obj for each key in path.

Parameters:
- **obj** (*(list|dict)*) – The top-level object.
- **path** (*(tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[str* (https://docs.python.org/3/library/stdtypes.html#str)*]|tuple[int* (https://docs.python.org/3/library/functions.html#int)*])*) – Keys to access parts of obj.

Returns: (*)

Example

```
>>> figure = {'data': [{'x': [5]}]}
>>> path = ('data', 0, 'x')
>>> get_by_path(figure, path)
[5]
```

plotly.utils. **node_generator**(*node*, *path=()*)

General, node-yielding generator.

Yields (node, path) tuples when it finds values that are dict instances.

A path is a sequence of hashable values that can be used as either keys to a mapping (dict) or indices to a sequence (list). A path is always wrt to some object. Given an object, a path explains how to get from the top level of that object to a nested value in the object.

Parameters:
- **node** (*(dict* (https://docs.python.org/3/library/stdtypes.html#dict)*)*) – Part of a dict to be traversed.
- **path** (*(tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[str* (https://docs.python.org/3/library/stdtypes.html#str)*])*) – Defines the path of the current node.

Returns: (Generator)

Example

```
>>> for node, path in node_generator({'a': {'b': 5}}):
...     print(node, path)
{'a': {'b': 5}} ()
{'b': 5} ('a',)
```

plotly.validator_cache module

class plotly.validator_cache. **ValidatorCache**

Bases: **object** (https://docs.python.org/3/library/functions.html#object)

static **get_validator** (*parent_path*, *prop_name*)

plotly.version module

plotly.version. **stable_semver** ()

Get the stable portion of the semantic version string (the first three numbers), without any of the trailing labels

‘3.0.0rc11’ -> ‘3.0.0’

plotly.widgets module