```
# Fill in student ID and name
#
student_id = "225297134"
student_first_last_name = "TRUONG VU NGUYEN"
print(student_id, student_first_last_name)
```

```
225297134 TRUONG VU NGUYEN
```

```
# install plotly and dash, if not yet already
! pip install plotly dash

import plotly, dash
print(plotly.__version__)
print(dash.__version__)
```

Hello world

Building and launching an app with Dash can be done with just 5 lines of code. Follow the tutorial (https://dash.plotly.com/tutorial) for more detail.

```
from dash import Dash, html

"""
Initialize the app.

This line is known as the Dash constructor and is responsible for
initializing your app.
It is almost always the same for any Dash app you create.
"""
app = Dash()


"""
App layout.

The app layout represents the app components that will be displayed in
the web browser and
here is provided as a list, though it could also be a Dash component.
In this example, a single component was added to the list: an
html.Div.
The Div has a few properties, such as children, which we use to add
text content to the page: "Hello World".
"""
app.layout = [
    html.Div(children='Hello World'),
    #*** Question: Add another html.Div to show your name, and re-run
the cell for output.
     html.Div(children='Truong Vu Nguyen')
]

if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")
```

```
Dash app running on http://127.0.0.1:8050/
```

```
<IPython.core.display.Javascript object>
```

Connecting to Data

There are many ways to add data to an app: APIs, external databases, local .txt files, JSON files, and more. In this example, we will highlight one of the most common ways of incorporating data from a CSV sheet.

```
# Import packages

# We import the dash_table module to display the data inside a Dash
DataTable.
from dash import Dash, html, dash_table

# We also import the pandas library to read the CSV sheet data.
import pandas as pd

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

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout.
# The 2nd item is a table with only 10 rows per page.
app.layout = [
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    #*** Question: Change page size and observe the change in widget
controls
    # such as, total number of pages.
     html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=20)
]
```

```
# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")
```

```
       country         pop continent  lifeExp     gdpPercap
0  Afghanistan  31889923.0      Asia   43.828    974.580338
1      Albania   3600523.0    Europe   76.423   5937.029526
2      Algeria  33333216.0    Africa   72.301   6223.367465
3       Angola  12420476.0    Africa   42.731   4797.231267
4    Argentina  40301927.0  Americas   75.320  12779.379640
Data rowsXcols: (142, 5)
Dash app running on http://127.0.0.1:8050/

<IPython.core.display.Javascript object>
```

Visualising data

The Plotly graphing library has more than 50 chart types to choose from. In this example, we will make use of the histogram chart.

```
# Import packages

# We import the dcc module (DCC stands for Dash Core Components).
# This module includes a Graph component called dcc.Graph, which is
used to render interactive graphs.
from dash import Dash, html, dash_table, dcc

# We also import the plotly.express library to build the interactive
graphs.
import plotly.express as px

import pandas as pd

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

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout
# 3rd component is an interactive graph (interaction no shown this
this example).
```

```
#
# Using the plotly.express library, we build the histogram chart
# and assign it to the figure property of the dcc.Graph. This displays
the histogram in our app.
#
app.layout = [
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp',
histfunc='avg')),
    #*** Question: Explore another histfunc other than 'avg' used
above and observe behaviour.
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=5),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp',
histfunc='sum'))
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")

       country         pop continent  lifeExp     gdpPercap
0  Afghanistan  31889923.0      Asia   43.828    974.580338
1      Albania   3600523.0    Europe   76.423   5937.029526
2      Algeria  33333216.0    Africa   72.301   6223.367465
3       Angola  12420476.0    Africa   42.731   4797.231267
4    Argentina  40301927.0  Americas   75.320  12779.379640
Data rowsXcols: (142, 5)
Dash app running on http://127.0.0.1:8050/

<IPython.core.display.Javascript object>
```

Controls and Callbacks

So far you have built a static app that displays tabular data and a graph. However, as you develop more sophisticated Dash apps, you will likely want to give the app user more freedom to interact with the app and explore the data in greater depth. To achieve that, you will need to add controls to the app by using the callback function.

In this example we will add radio buttons to the app layout. Then, we will build the callback to create the interaction between the radio buttons and the histogram chart.

```
# Import packages

# We import dcc like we did in the previous section to use dcc.Graph.
# In this example, we need dcc for dcc.Graph as well as the radio
buttons component, dcc.RadioItems.
```

```
#
# To work with the callback in a Dash app, we import the callback
module and the two arguments
# commonly used within the callback: Output and Input.
#
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px

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

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'],
value='lifeExp', id='controls-and-radio-item'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
    #*** Question: Use line graphs instead of histogram.
]

#
# Notice that we add that RadioItems component to the layout, directly
above the DataTable.
# There are three options, one for every radio button.
# The lifeExp option is assigned to the value property, making it the
currently selected value.
#
# Both the RadioItems and the Graph components were given id names:
these will be used by
# the callback to identify the components.
#
# The inputs and outputs of our app are the properties of a particular
component.
# In this example, our input is the value property of the component
that has the ID "controls-and-radio-item".
# If you look back at the layout, you will see that this is currently
lifeExp.
# Our output is the figure property of the component with the ID
"controls-and-graph", which
# is currently an empty dictionary (empty graph).
#
# The callback function's argument col_chosen refers to the component
```

```
property of the input lifeExp.
# We build the histogram chart inside the callback function, assigning
the chosen radio item to the
# y-axis attribute of the histogram. This means that every time the
user selects a new radio item,
# the figure is rebuilt and the y-axis of the figure is updated.
#
# Finally, we return the histogram at the end of the function. This
assigns the histogram to the figure
# property of the dcc.Graph, thus displaying the figure in the app.
#

# Add controls to build the interaction
@app.callback(
    Output('controls-and-graph', 'figure'),
    Input('controls-and-radio-item', 'value')
)
def update_graph(col_chosen):
    # ▯ Line graph instead of histogram
    fig = px.line(
        df.sort_values("country"),  # sort for a cleaner line
        x="country",
        y=col_chosen,
        markers=True,
        title=f"{col_chosen} by Country"
    )
    return fig

@callback(
    Output(component_id='controls-and-graph',
component_property='figure'),
    Input(component_id='controls-and-radio-item',
component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen,
histfunc='avg')
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")
```

```
----------------------------------------------------------------------
-----
DuplicateCallback                         Traceback (most recent call
last)
DuplicateCallback: The callback `controls-and-graph.figure` provided
with `dash.callback` was already assigned with `app.callback`.
```

```
Dash app running on http://127.0.0.1:8050/

<IPython.core.display.Javascript object>
```