Конспект по теме "Проектирование и разработка дашбордов в dash"

Дашборды

Дашборд — автоматически обновляемый интерактивный отчёт, отображающий набор бизнес-показателей для управления компанией:

- «автоматически обновляемый» — данные, по которым строится дашборд постоянно обновляются;
- «интерактивный» — на дашборде обычно размещают элементы управления для фильтрации отображаемой информации;
- «набор бизнес-показателей» — на дашборде отображают информацию для решения бизнес-задач;
- «для управления компанией» — дашборд применяют для принятия бизнес-решений.

На практике источником информации для дашбордов становятся типовые отчёты, когда их формируют с постоянной периодичностью. Чтобы не платить за коммерческие системы и получить неограниченные возможности построения дашбордов, будем создавать их в библиотеке **dash** в *Python*. **dash** — набор Python-библиотек, позволяющих конструировать дашборды и отображать их в веб-браузере. За создание дашбордов отвечают библиотеки *dash* и *plotly*. А за отображение в браузере — микрофреймворк **Flask** — набор базовых библиотек для создания веб-приложений.

Простейший пример дашборда, построенного в *dash*:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import dash
import dash_core_components as dcc
import dash_html_components as html

import plotly.graph_objs as go

import pandas as pd

# задаем лейаут (макет)
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app.layout = html.Div(children=[

    # формируем заголовок
    html.H1(children = 'Линейная функция'),

    # график
```

```
    dcc.Graph(
        figure = {
                'data': [go.Scatter(x = pd.Series(range(-100, 100, 1)),
                                    y = pd.Series(range(-100, 100, 1)),
                                    mode = 'lines',
                                    name = 'linear_func')],
                'layout': go.Layout(xaxis = {'title': 'x'},
                                    yaxis = {'title': 'y'})
             },
        id = 'linear_func_id'
    ),

])

# описываем логику дашборда
if __name__ == '__main__':
    app.run_server(debug=True)
```

Лейаут дашборда — графическая часть, которая отобразит все графики и его элементы управления на веб-странице.

Как запустить дашборд на локальной машине

Как запустить дашборд на локальной машине в командной строке Linux:

1. Откройте командную строку Linux;
2. Создайте простой дашборд и сохраните его в файл `cloud_test.py`.

3. Установите dash. Для этого сначала нужно развернуть **pip** — приложение для установки пакетов Python. Последовательно выполните в командной строке:

```
sudo apt update
sudo apt install python3-pip # отвечайте yes на все вопросы
```

Первая команда обновляет библиотеку доступных приложений. В Linux приложения устанавливают в библиотеке `apt`. По умолчанию `pip` не входит в библиотеку `apt` , поэтому её нужно обновить.

Вторая команда — `python3-pip` — устанавливает `pip` для *Python3*.

Команда `sudo` означает, что последующие команды должны быть выполнены от лица системного пользователя (администратора). При выполнении команды `sudo` потребуется ввести пароль вашего пользователя. Обычно он совпадает с паролем на вход в систему.

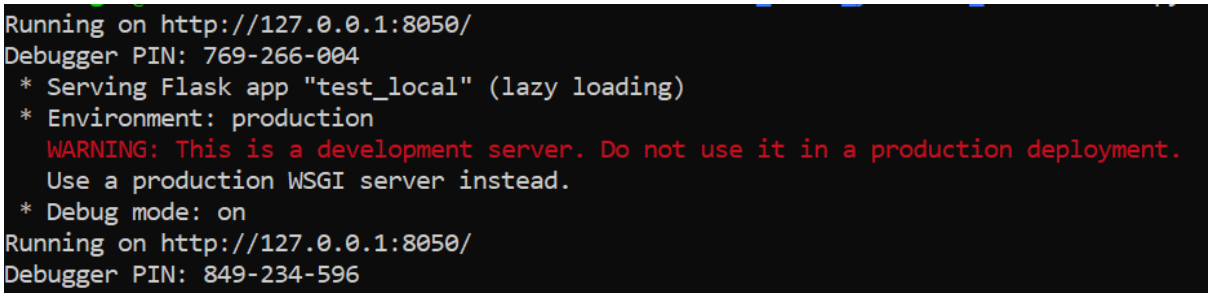
Выполните команду установки dash:

```
pip3 install dash==1.4.1
```

После этого, перейдите в папку, в которой вы сохранили `cloud_test.py` и выполните команду:

```
python3 cloud_test.py
```

Система отобразит такой результат:



Скопируйте строку `http://127.0.0.1:8050/` и откройте эту ссылку в браузере. Увидите локальную версию вашего первого дашборда.

Запуск дашборда на виртуальной машине сложнее и требует знания, как устанавливать и настраивать веб-сервера. Чтобы разворачивать дашборды на вашей будущей работе, нужно будет пообщаться с системными администраторами и сказать им волшебную фразу «Мне нужно развернуть Flask-приложение на Apache-сервере».

Сбор требований при создании дашборда

Для построения дашбордов нужно общаться с заказчиками дашбордов, собирать функциональные требования и на их основе составлять **технические задания**, или **ТЗ**.

При построении дашбордов следует выяснить у заказчика детали:

- Какую бизнес-задачу предполагается решать с помощью дашборда, кто будет основным пользователем дашборда;
- Насколько часто предполагается пользоваться дашбордом;
- Состав данных для дашборда: какие метрики (KPI) на нём должны отображаться, по каким параметрам данные должны группироваться, на основе каких параметров должны формироваться когорты пользователей;
- Характер отображаемых на дашборде данных (абсолютные или относительные значения или и те и другие);
- Источники данных для дашборда;
- Базу данных, в которой будут хранится агрегированные данные;
- Частоту обновления данных;
- Какие графики должны на нём отображаться и в каком порядке;
- Какие элементы управления должны быть на дашборде.

Если ТЗ вам не предоставили другие специалисты компании, его нужно составить самостоятельно.

Прежде всего, следует выяснить, какую именно бизнес-задачу должен решать дашборд. На этом шаге нужно ответить на два вопроса:

1. Имеет ли смысл строить дашборд для решения задачи?

Дашборд стоит строить, когда заказчик будет то и дело обращаться к нему для принятия решений. Если заказчик планирует изучить состояние дел единожды, лучше составить обычный отчёт.

2. Сколько нужно дашбордов?

Определить, нужно ли разделить поставленную задачу на несколько дашбордов, довольно сложно. Придётся полагаться на опыт и здравый смысл. Если вы видите, что в требованиях заказчика явно смешиваются несколько бизнес-задач, или нужно много разнородных данных, скорее всего, придётся делать несколько.

Затем выясните, какие нужны данные. На этом же шаге установите, какие измерения нужно добавить к данным дашборда.

Затем обсудите с заказчиком, какие графики он хочет видеть на дашборде. Здесь нужно понять, какие данные важнее всего для заказчика. Чем важнее график, тем бóльшую площадь дашборда он должен занимать. Для каждого графика узнайте, должен ли он показывать относительные или абсолютные величины.

Здесь же выясните, какие элементы управления и фильтры нужно добавить на дашборд. Практически все дашборды требуют фильтрации по времени. Остальные фильтры и элементы управления почти всегда повторяют измерения данных, которые требуются для дашборда.

Вместе с заказчиком набросайте **макет дашборда** — карту, эскиз, на котором показаны типы графиков, их относительные размеры и взаимное расположение. Для макета постройте простую таблицу

Результатом общения с заказчиком должно стать техническое задание. Это может быть формальный документ, заметка или презентация.

Когда макет и ТЗ готовы, перейдите к технической части проектирования. На этом этапе выясните у администраторов баз данных, какие источники вы можете использовать. Решите, в какой БД будут храниться агрегированные данные, и на каком сервере будет работать скрипт дашборда.

Вот базовые элементы, которые включает в себя каждый дашборд:

- **Заголовок** — должен рассказать пользователю, что именно отображает дашборд;
- **Описание дашборда** — короткий текст, описывающий задачу, которую решает дашборд, и особенности логики его работы (если есть);
- **Графики и диаграммы;**
- **Элементы управления**.

Как создавать основные типы графиков в dash

Импортируем библиотеку `dash_html_components` как `html` :

```
import dash_html_components as html
```

Эта конструкция добавляет HTML-тег H1 (заголовок первого уровня), в котором указывают название дашборда:

```
html.H1(children = 'Заголовок самого лучшего дашборда от самого лучшего аналитика')
```

Если нужно добавить на дашборд картинку, размеченную тегом <img>, делают так:

```
html.Img(src='адрес_картинки')
```

А если нужен текст, например, пояснение о функциональности дашборда или заголовок элемента управления (в разметке тег <label>), то следует `html.Label()`:

```
html.Label('Тут ваш текст')
```

Изучите компоненты библиотеки `dash_html_components` самостоятельно: https://dash.plot.ly/dash-html-components.

Импортируем библиотеку `dash_core_components` как `dcc`. Компоненты этой библиотеки ответственны за отображение графиков и элементов управления дашборда:

```
import dash_core_components as dcc
```

В компоненте `dcc.Graph()` добавляют основной график дашборда:

```
dcc.Graph(
        figure = {
                'data': [go.Bar(x = max_urbanization['Entity'],
                                y = max_urbanization['Urban'],
                                name = 'max_urbanization')],
                'layout': go.Layout(xaxis = {'title': 'Страна'},
                                    yaxis = {'title': 'Макс. % городского населения'})
             },
        id = 'urbanization_by_year'
    ),
```

У этого компонента два параметра:

- `figure` описывает, какой именно график будет показан;
- `id` — уникальное имя графика, которое вы задаёте сами. На основе этого параметра настраивается интерактивность дашборда.

Рассмотрим, что происходит внутри `figure`:

```
figure = {
                'data': [go.Bar(x = max_urbanization['Entity'],
                                y = max_urbanization['Urban'],
                                name = 'max_urbanization')],
                'layout': go.Layout(xaxis = {'title': 'Страна'},
                                    yaxis = {'title': 'Макс. % городского населения'})
            },
```

У `figure` два свойства:

- `'data'` задаёт набор графиков, которые будут отображены `dcc.Graph`. На дашборд можно добавить диаграмму любого вида, входящую в библиотеку *Plotly*.
- `'layout'` задаёт параметры отображения графиков. Здесь указываем компонент `go.Layout` для отображения подписей осей.

Содержание `'data'` задаётся динамически: можно сформировать набор графиков в любом месте программы и передать его параметру `'data'`.

Изучим, как можно нарисовать базовые типы графиков в библиотеке `plotly.graph_objs`.

go.Scatter рисует:

- линейные графики (*line*),
- области с накоплением (*stacked area*)
- точечные диаграммы (*scatter-plot*).

Основные параметры `go.Scatter`:

- x, y — объекты pd.Series (например, столбец датафрейма), содержащие значения по оси x и y;
- mode — режим отображения. Например, `'markers'` выведет `scatter-plot`; `'lines'` — линии; `'lines'` с параметром `stackgroup = 'one'`, выведет график областей с накоплением.

go.Bar рисует столбчатую диаграмму (*bar plot*). Основные параметры `go.Bar`:

- x, y — объекты pd.Series (например, столбец датафрейма), содержащие значения по оси x и y.
- **barmode** определяет, каким образом на одном графике будут отображаться два и более набора данных:
  - `barmode = 'stack'` выведет один набор данных над другим (*stacked bar chart*);
  - `barmode = 'group'` выведет наборы данных рядом.

Заметьте, что `barmode` указывают в секции `layout`.

go.Pie рисует круговую диаграмму (*pie chart*). Основные параметры go.Pie:

- `labels` — объект pd.Series, содержащий названия категорий;
- `values` – объект pd.Series, содержащий значения категорий.

go.Box рисует диаграммы размаха (box plot). Основные параметры go.Box:

- y — объект pd.Series, содержащий значения нужной переменной.

go.Table рисует таблицы. Основные параметры go.Table:

- `header` — словарь (*dict*) заголовков столбцов. Значения заголовков передаются в виде массива в параметре `values`. Например:

```
header = {'values': ['<b>Столбец 1</b>', '<b>Столбец 2</b>']}
```

`<b>` здесь — тег HTML, задающий жирное начертание текста.

- `cells` — словарь (*dict*) ячеек таблицы. Значения передаются в виде массива в параметре `values`. Чтобы удобно передать в `cells` значения датафрейма, его **траспонируют,** или «переворачивают» оператором `T`.

```
cells = {'values': your_data_frame.T.values}
```

Изучим транспонацию на простом примере. Есть датафрейм *df*:

	Яблоки	Бананы	Сыр	Груши
Вася	1	5	8	3
Оля	10	19	22	21
Петя	45	34	99	44

Его транспонированная версия (`df.T`) будет выглядеть так:

	Вася	Оля	Петя
Яблоки	1	10	45
Бананы	5	19	34
Сыр	8	22	99
Груши	3	21	44

Пример работы с таблицей:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

import plotly.graph_objs as go

from datetime import datetime

import pandas as pd

# получаем данные и трансформируем их
urbanization = pd.read_csv('data/urbanization.csv')
urbanization['Year'] = pd.to_datetime(urbanization['Year'], format = '%Y')

urbanization_table = urbanization.copy()
urbanization_table['Year'] = urbanization_table['Year'].dt.date
urbanization_table['Urban'] = urbanization_table['Urban'].round(2)

# задаём лейаут
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app.layout = html.Div(children=[

    # таблица урбанизации
    dcc.Graph(
        figure = {
                'data': [go.Table(header = {'values': ['<b>Страна</b>',
                                                       '<b>Год</b>',
                                                       '<b>% городского населения</b>'],
                                            'fill_color': 'lightgrey',
                                            'align': 'center'
                                            },
                                  cells = {'values': urbanization_table.T.values})],
                'layout': go.Layout(xaxis = {'title': 'Страна'},
                                    yaxis = {'title': 'Макс. % городского населения'})
             },
        id = 'urbanization_table'
    ),

])

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

Получим такую таблицу:

Страна	Год	% городского населения
Afghanistan	1950-01-01	6
Africa	1950-01-01	14.28
Albania	1950-01-01	20.53
Algeria	1950-01-01	22.21
American Samoa	1950-01-01	61.77
Andorra	1950-01-01	38.8
Angola	1950-01-01	7.58
Anguilla	1950-01-01	100
Antigua and Barbuda	1950-01-01	30.06
Argentina	1950-01-01	65.34
Armenia	1950-01-01	40.34
Aruba	1950-01-01	50.93

Кроме базовых параметров компоненты имеют массу других настроек (цвет линий и т.д.). Изучите, как работают эти свойства на примерах: https://plot.ly/python/.

Для вдохновения ознакомьтесь с галереей визуализаций, построенных на dash: https://dash-gallery.plotly.host/Portal/.

Основы работы с элементами управления

Любой дашборд должен быть **интерактивным** — иметь элементы управления, позволяющие фильтровать отображаемые данные.

Посмотрим на код дашборда с элементом управления:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import dash
import dash_core_components as dcc
import dash_html_components as html

import plotly.graph_objs as go

from datetime import datetime

import pandas as pd

# получаем данные и трансформируем их
urbanization = pd.read_csv('data/urbanization.csv')
urbanization['Year'] = pd.to_datetime(urbanization['Year'], format = '%Y')
max_urbanization = (urbanization.groupby('Entity')
                                .agg({'Urban': 'max'})
                                .reset_index()
                                .sort_values(by = 'Urban', ascending = False)
                                .head(25))

# задаём лейаут
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app.layout = html.Div(children=[

    # формируем заголовок тегом HTML
    html.H1(children = 'Максимальная урбанизация стран мира, топ-25'),

    # выбор временного периода
    html.Label('Временной период:'),
    dcc.DatePickerRange(
        start_date = urbanization['Year'].dt.date.min(),
        end_date = urbanization['Year'].dt.date.max(),
        display_format = 'YYYY',
        id = 'dt_selector',
    ),

    # график урбанизации
    dcc.Graph(
        figure = {
                'data': [go.Bar(x = max_urbanization['Entity'],
                                y = max_urbanization['Urban'],
                                name = 'max_urbanization')],
                'layout': go.Layout(xaxis = {'title': 'Страна'},
                                    yaxis = {'title': 'Макс. % городского населения'})
            },
        id = 'urbanization_by_year'
```

```
    ),

])

# описываем логику дашборда

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

И, наконец, сам элемент управления:

```
# выбор временного периода
    html.Label('Временной период:'),
    dcc.DatePickerRange(
        start_date = urbanization['Year'].dt.date.min(),
        end_date = urbanization['Year'].dt.date.max(),
        display_format = 'YYYY',
        id = 'dt_selector',
    ),
```

Здесь `html.Label` задаёт название элемента управления. `dcc.DatePickerRange` — сам элемент управления — диалог выбора даты начала и даты окончания временного интервала. Элемент управления `dcc.DatePickerRange` имеет 4 параметра:

- `start_date` — дата начала. Она по умолчанию устанавливается равной минимальному (первому) году наблюдений: `urbanization['Year'].dt.date.min()`;
- `end_date` — дата окончания. Она по умолчанию равна максимальному (последнему) году наблюдений: `urbanization['Year'].dt.date.max()`;
- `display_format` — формат отображения даты и времени;
- `id` — уникальный идентификатор элемента управления.

Задать элемент управления просто: достаточно добавить в лейаут нужный объект из библиотеки dash, указать его параметры и id. Обратите внимание, что параметры элемента управления формируются динамически на основе исходных данных: в `start_date` и `end_date` записываются минимальное и максимальное значение дат датафрейма `urbanization`.

Базовые элементы управления в dash

Вот самые популярные элементы управления в dash:

- **Диалог выбора диапазонов да**т (*date and time range selector*) — выбор временного промежутка, для которого отображаются данные;
- **Набор флажков** (*checkboxes*) — выбор категорий данных;
- **Выпадающий список** (*dropdown*) — ещё один элемент управления для выбора категорий данных;
- **Радиокнопка** (*radio button*) — выбор единственной опции из набора;

Диалог выбора диапазонов дат применяют для выбора дат и времени начала и окончания временных интервалов:

```
dcc.DatePickerRange(start_date = '2016',
                    end_date = '2019',
                    display_format = 'YYYY',
                    id = 'dt_selector', )
```

Здесь `start_date`, `end_date` — строки в формате, указанном в параметре `display_format`.

Дополнительная документация по диалогу выбора диапазона дат: https://dash.plot.ly/dash-core-components/datepickerrange

Набор флажков применяют, когда нужно обеспечить одновременный выбор одной или нескольких категорий значений:

```
dcc.Checklist( options = [{'label': 'Африка', 'value': 'afr'},
                          {'label': 'Евразия', 'value': 'eur'},
                          {'label': 'Австралия', 'value': 'au'},
                          {'label': 'Америка', 'value': 'am'}],
               value = ['afr', 'eur', 'au', 'am'],
               id = 'continent_selector' )
```

В параметре `options` задают массив опций выбора. Для каждой опции `label` — значение, которое видит пользователь, `value` — техническое значение для обработки внутри программы. В параметре `value` указывают набор `value`, выбранных по умолчанию.

Дополнительная документация по диалогу выбора набора флажков: https://dash.plot.ly/dash-core-components/checklist

Выпадающий список применяют, когда нужно обеспечить одновременный выбор одного или нескольких значений категорий:

```
dcc.Dropdown( options = [{'label': 'Африка', 'value': 'afr'},
                         {'label': 'Евразия', 'value': 'eur'},
                         {'label': 'Австралия', 'value': 'au'},
                         {'label': 'Америка', 'value': 'am'}],
              value = ['afr', 'eur', 'au', 'am'],
              multi = True, id = 'continent_selector' )
```

Все параметры аналогичны набору флагов.

Параметр `multi` управляет возможностью выбрать несколько элементов из списка.

Дополнительная документация по выпадающим спискам: https://dash.plot.ly/dash-core-components/dropdown

Радиокнопку применяют для выбора единственного варианта из предложенного набора.

Вот код:

```
dcc.RadioItems(options = [{'label': 'Чай', 'value': 'tea',
                           {'label': 'Кофе', 'value': 'coffee'},
                           {'label': 'Потанцуем', 'value': 'lets_dance'},
                          ],
               value = 'tea',
               id = 'drink_selector')
```

В параметре `options` задают массив опций для выбора. Для каждой опции `label` — значение, которое видит пользователь, `value` — техническое значение для обработки внутри программы. В параметре `value` указывают набор `value`, выбранных по умолчанию.

Изучите дополнительную документацию по радиокнопкам: https://dash.plot.ly/dash-core-components/radioitems

Другие типы элементов управления: https://dash.plot.ly/dash-core-components

Элементы управления и интерактивность

В дашборде, код которого представлен нижк, один элемент управления — радиокнопка, позволяющая выбрать отображаемую тригонометрическую функцию, или режим отображения. Вот код дашборда:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objs as go

import pandas as pd

from dash.dependencies import Input, Output
import math


# задаём лейаут
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app.layout = html.Div(children=[

    # формируем заголовок тегом HTML
    html.H1(children = 'Тригонометрические функции'),

    # выбор режима отображения
    html.Label('Режим отображения:'),
    dcc.RadioItems(
        options = [
            {'label': 'Показать sin(x)', 'value': 'sin'},
```

```
            {'label': 'Показать cos(x)', 'value': 'cos'},
        ],
        value = 'sin',
        id = 'mode_selector'
    ),

    # график
    dcc.Graph(
        # параметр figure определяется динамически
        id = 'trig_func'
    ),
])

# описываем логику дашборда
@app.callback(
    [Output('trig_func', 'figure'),
    ],
    [Input('mode_selector', 'value'),
    ])
def update_figures(selected_mode):

  # формируем графики для отрисовки с учетом фильтров
  x = range(-100, 100, 1)
  x = [x / 10 for x in x]
  y = [math.sin(x) for x in x]
  if selected_mode == 'cos':
    y = [math.cos(x) for x in x]
  data = [
          go.Scatter(x = pd.Series(x), y = pd.Series(y), mode = 'lines', name = 'sin(x)'),
         ]

  # формируем результат для отображения
  return (
            {
                'data': data,
                'layout': go.Layout(xaxis = {'title': 'x'},
                                    yaxis = {'title': 'y'})
             },
  )

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

Импортируем компоненты библиотеки dash, отвечающие за сигналы управления дашбордом. Сигнал управления генерируется каждый раз, когда пользователь меняет какой-то из элементов управления:

```
from dash.dependencies import Input, Output
```

В лейауте кроме заголовка есть два элемента. Первый из них:

```
html.Label('Режим отображения:'),
    dcc.RadioItems(
        options = [
            {'label': 'Показать sin(x)', 'value': 'sin'},
            {'label': 'Показать cos(x)', 'value': 'cos'},
        ],
```

```
        value = 'sin',
        id = 'mode_selector'
    ),
```

задаёт радиокнопку для выбора режима отображения. Второй:

```
# график
    dcc.Graph(
        # параметр figure определяется динамически
        id = 'trig_func'
    ),
```

...задаёт график, который будет отображаться на дашборде. Сейчас у этого графика нет никакой отображаемой части: отсутствует параметр `figure`. Все потому что дашборд будет формировать `figure` динамически, в зависимости от значений, выбранных пользователем в элементах управления.

Сразу за лейаутом идёт странный блок кода:

```
# описываем логику дашборда
@app.callback(
    [Output('trig_func', 'figure'),
    ],
    [Input('mode_selector', 'value'),
    ])
def update_figures(selected_mode):
    # код обновления графика
```

Это функция обработки элементов управления. Прежде чем разберём её подробно, изучим теорию.

Раньше вы работали только с Python-кодом, который выполнялся последовательно. Такой тип выполнения команд называется **синхронным**. dash работает по другой модели программирования — **асинхронной**. В ней одна часть программы генерирует сигналы, а другие части эти сигналы ловят. В зависимости от пойманного сигнала, выполняют те или иные действия.

```
# описываем логику дашборда
@app.callback(
    [Output('trig_func', 'figure'),
    ],
    [Input('mode_selector', 'value'),
    ])
def update_figures(selected_mode):

  # формируем графики для отрисовки с учетом фильтров
  if selected_mode == 'cos':
    y = [math.cos(x) for x in x]
  data = [
          go.Scatter(x = pd.Series(x), y = pd.Series(y), mode = 'lines', name = 'sin(x)'),
         ]

  # формируем результат для отображения
  return (
             {
                 'data': data,
```

```
                'layout': go.Layout(xaxis = {'title': 'x'},
                                    yaxis = {'title': 'y'})
            },
  )
```

Это **декоратор** — ещё одна конструкция Python:

```
@app.callback(
    [Output('trig_func', 'figure'),
    ],
    [Input('mode_selector', 'value'),
    ])
```

С практической точки зрения конструкция:

```
@app.callback(
    [Output('trig_func', 'figure'),
    ],
    [Input('mode_selector', 'value'),
    ])
def update_figures(selected_mode):
…
```

говорит функции `update_figures`, от каких элементов управления ждать сигналы (`Input`) и какие элементы дашборда нужно обновить после выполнения функции (`Output`). В примере `update_figures`:

- Ждёт сигнал от радиокнопки с `id = 'mode_selector'`. Получив сигнал, `update_figures` возьмёт из него параметр 'value';
- После выполнения функция `update_figures` передаст свой результат в параметр `'figure'` графика `'trig_func'`. Так и происходит динамическое обновление графика на дашборде.

Здесь входной параметр `selected_mode` получит своё значение из `Input('mode_selector', 'value')`. Так, `selected_mode` будет равен `value` элемента управления с `id = 'mode_selector'`.

Что делать, если несколько элементов управления генерируют сигналы для нескольких графиков? Например, вот такой случай:

```
@app.callback(
    [Output('plot_1', 'figure'),
     Output('plot_2', 'figure')
    ],
    [Input('control_1', 'value'),
     Input('control_2', 'value'),
     Input('control_3', 'value')
    ])
```

На дашборде три элемента управления с идентификаторами: `control_1`, `control_2`, `control_3` и два графика с идентификаторами: `plot_1` и `plot_2`. `update_figures` должна быть определена вот так:

```
def update_figures(control_1_value, control_2_value, control_3_value):
# формируем динамические графики
return (
            { # это график для plot_1
                'data': plot_1_data,
                'layout': go.Layout(…)
})
             },
            { # это график для plot_2
                'data': plot_2_data,
                'layout': go.Layout(…)
           },
  )
```

Порядок входных и выходных параметров определяется порядком заданных элементов `Input` и `Output` в декораторе.

Элементы дашборда

Чтобы размещать элементы дашборда в соответствии с макетом, нужно научиться работе с HTML — вёрстке.

Из всех HTML-тегов нас интересует ** **. В своей самой простой форме этот тег логически разделяет содержимое веб-страницы на блоки.

Каждый дашборд на dash — тоже веб-страница. dash отображает элементы дашбордов во фреймворке **Bootstrap** (https://www.w3schools.com/bootstrap/default.asp). **CSS** — каскадные таблицы стилей — язык описания стилей отображения HTML-элементов на странице.

HTML-элементам задаются классы — особые имена. Всем разбросанным по странице элементам с одним классом можно прописать общее правило CSS, так что эти элементы будут отображаться одинаково. Нам интересно задать классы, которые помогут отображать элементы `div` таблицей, выстраивать их в ряды и столбцы.

Для этого понадобятся классы `'row'` (в ряд) и `'N columns'` (в N колонок). Для этих классов написана CSS-библиотека dash. В имени `'N columns'` буква N заменяется на английское числительное, обозначающее число колонок. Например, `'three columns'` — три колонки.

Чтобы понять, как работают классы, взглянем на простой пример. HTML-код:

```
<div>
  <div>Раз</div>
  <div>Два</div>
  <div>Три</div>
</div>
```

...отобразит в браузере вот такой результат:

```
Раз
Два
Три
```

Если применить классы dash:

```
<div class = 'row'>
  <div class = 'four columns' >Раз</div>
  <div class = 'four columns' >Два</div>
  <div class = 'four columns' >Три</div>
</div>
```

В браузере получим:

```
Раз Два Три
```

Три элемента *div* с классом `'four columns'` расставлены в ряд, потому что оказались внутри элемента *div* с классом `row`. Так элементы div выстраивают в ряды и колонки.

Особенность работы с классами в dash — ограничение на число колонок в ряду. Их не может быть больше 12.

Подробнее о том, как работать с классами Bootstrap можно почитать здесь: https://www.w3schools.com/bootstrap/bootstrap_grid_basic.asp

Узнаем, как задавать `div` с нужными классами в dash. Это можно сделать в элементе `html.Div`. Напишем код примера с двумя колонками:

```
html.Div([
  html.Div([
    html.Label('Я шириной 9 колонок из 12-ти возможных:'),
  ], className = 'nine columns'),
  html.Div([
    html.Label('Я шириной 3 колонки из 12-ти возможных:'),
  ], className = 'three columns'),
], className = 'row'),
```

Итак, внутри лейаута дашборда в элементах `html.Div` можно задавать HTML-теги `<div>`. А в параметре `className` указывать им нужные классы и управлять шириной отображаемых элементов.

Узнаем, как регулировать высоту элементов. Высота `div` зависит от высоты его содержимого. Поэтому придётся устанавливать высоту графиков, которые «вкладываются» в `div`. Её настраивают в параметре `style`. Высота может быть установлена в разных единицах измерения, но удобнее всего пользоваться относительными — процентами от ширины страницы — **vw**. Например, если для графика нужно установить высоту в 25% от ширины страницы, dash-код будет выглядеть вот так:

```
dcc.Graph(
  style = {'height': '25vw'},
  id = 'sales_by_platform'
)
```

Осталось познакомиться с дополнительными элементами dash, которые помогут сделать дашборд более информативным и читаемым.

Элемент **html.H1** выводит заголовок дашборда. Вот пример его использования:

```
html.H1(children = 'Я заголовок')
```

Элемент **html.Label** выводит текст:

```
html.Label('Съешь ещё этих мягких французских булок да выпей чаю.')
```

Элемент **html.Br** позволяет добавить пустую строку между двумя элементами дашборда, если он кажется слишком скученным.

```
html.Br()
```