POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Guide TP2

Version Python

Technologies

For the labs, we are going to use:

- HTML / CSS
- Plotly + Dash

Often, the TPs can be accomplished many different ways - we expect you to choose the option that uses these technologies!

Technologies

HTML

Hypertext Markup Language is used to structure content for web browsers

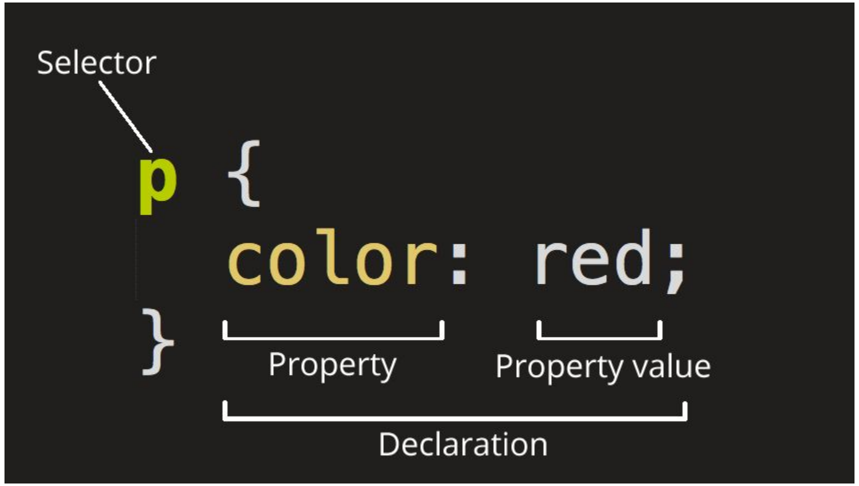
```
<!DOCTYPE html>
<html>
        <head>
                <title>Page Title</title>
        </head>
        <body>
                <h1>Page Title</h1>
                <p>This is a really interesting paragraph.</p>
        </body>
</html>
```

- <...> open, </…> close
- DOM - Document Object Model (hierarchical structure of HTML)
  - <...> </…> is an element
  - Relationships:
    - Siblings (h1 and p are siblings)
    - Childrens (h1 and p)
    - Parent (body)
    - Descendant (html)

Technologies

CSS

- Describes the presentation of an HTML page
- Uses **selectors** and **declarations**



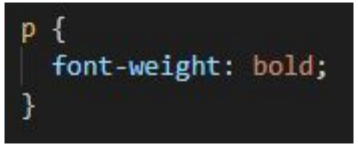
```
p {
    color: red;
}
```

CSS (source)

Technologies

CSS selectors and declarations

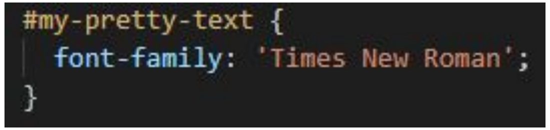
- Examples of selectors
  - By type of element



```
p {
  font-weight: bold;
}
```

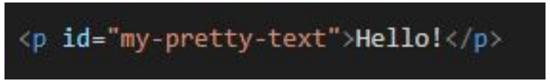
All <p> elements will be bold

  - On a single element : **by id**



```
#my-pretty-text {
  font-family: 'Times New Roman';
}
```

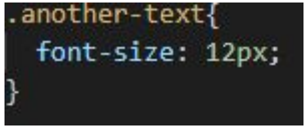
Font of element with id “my-pretty-text” will be “Times New Roman”



```
<p id="my-pretty-text">Hello!</p>
```

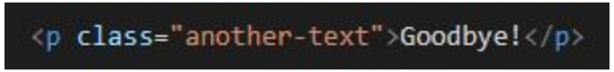
Corresponding HTML element

  - On a group of elements : **by class**



```
.another-text{
  font-size: 12px;
}
```

Font of elements with class “another-text” will be 12px



```
<p class="another-text">Goodbye!</p>
```

Corresponding HTML element

Technologies

CSS selectors and declarations

- Declarations
  - Depend on the style attributes of the selected element(s)

- Examples…

  - **width and height** : width and height of an element
  - **fill** : color of an element
  - **margin** : margin around an element
  - **font-size** : size of the font of text element

Technologies

Plotly et Dash

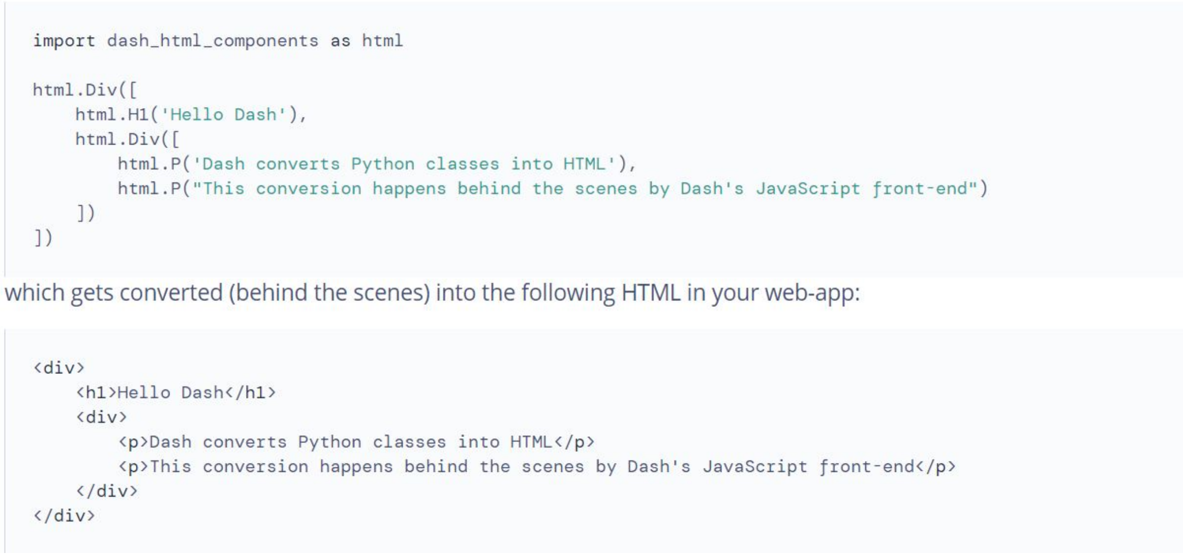
- Plotly is a Python graphing library that makes interactive, publication-quality graphs.

- Dash is a web application framework that provides pure Python abstraction around HTML, CSS, and JavaScript.

Technologies

Dash

- Instead of writing HTML or using an HTML templating engine, you compose your layout using Python structures with the ***dash-html-components*** library.



```
import dash_html_components as html

html.Div([
    html.H1('Hello Dash'),
    html.Div([
        html.P('Dash converts Python classes into HTML'),
        html.P("This conversion happens behind the scenes by Dash's JavaScript front-end")
    ])
])
```

which gets converted (behind the scenes) into the following HTML in your web-app:

```
<div>
    <h1>Hello Dash</h1>
    <div>
        <p>Dash converts Python classes into HTML</p>
        <p>This conversion happens behind the scenes by Dash's JavaScript front-end</p>
    </div>
</div>
```

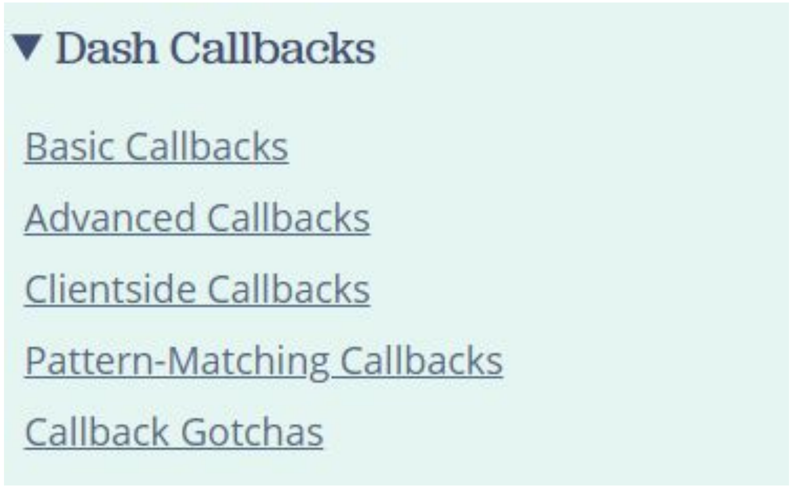
You can find all the HTML components on the link below:

Dash HTML Components | Dash for Python Documentation | Plotly

Callbacks

How to make your chart interactive

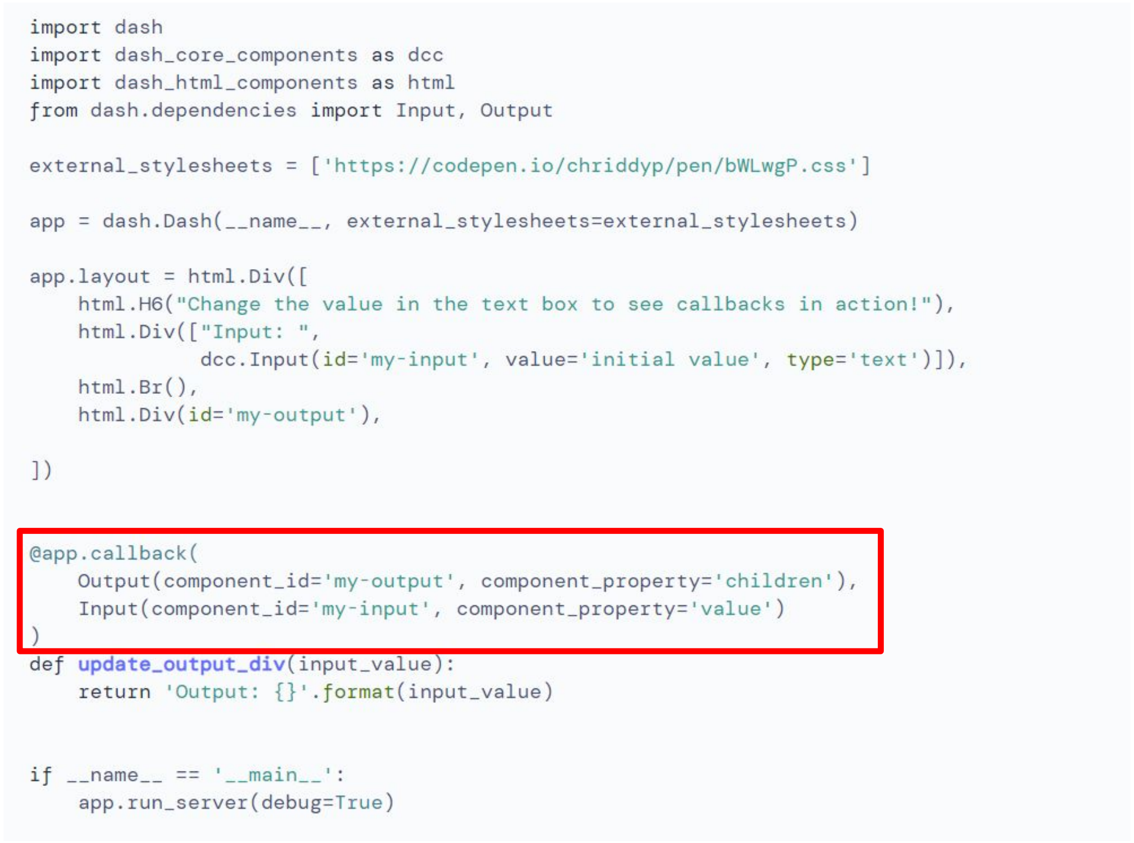
- Callback functions: Python functions that are automatically called by Dash whenever an input component's property changes.



Callbacks

Simple Interactive Dash App

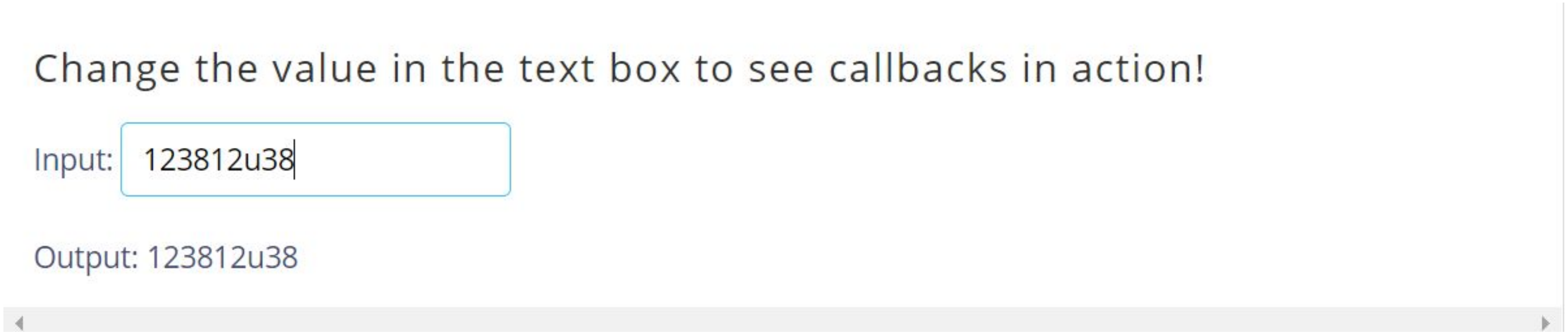
- You will need a component ***ID*** and a component ***property*** to inform to the app callback the actions on your ***Input*** and ***Output***



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

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),

])

@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)


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

Callbacks

Simple Interactive Dash App


Change the value in the text box to see callbacks in action!

Input: 123812u38

Output: 123812u38

Callbacks

Simple Interactive Dash App

- In Dash, the inputs and outputs of our application are simply the properties of a particular component. In this example, our input is the "value" property of the component that has the ID "my-input". Our output is the "children" property of the component with the ID "my-output"
- The component_id and component_property keywords are optional (there are only two arguments for each of those objects). They are included in this example for clarity but will be omitted in the rest of the documentation for the sake of brevity and readability.
- Notice how we don't set a value for the children property of the my-output component in the layout. When the Dash app starts, it automatically calls all of the callbacks with the initial values of the input components in order to populate the initial state of the output components. In this example, if you specified something like html.Div(id='my-output', children='Hello world'), it would get overwritten when the app starts.

Callbacks

Dash app with State

- In some cases, you might have a "form"-type pattern in your application. In such a situation, you might want to read the value of the input component, but only when the user is finished entering all of his or her information in the form.

- ***State*** allows you to pass along extra values without firing the callbacks.

Callbacks

Simple Interactive Dash App

- Check for more information:
    - https://dash.plotly.com/basic-callbacks
    - https://www.youtube.com/watch?v=mTsZL-VmRVE
    - https://medium.com/@benshentist/dash-callbacks-where-the-magic-happens-ab19260dbc7e

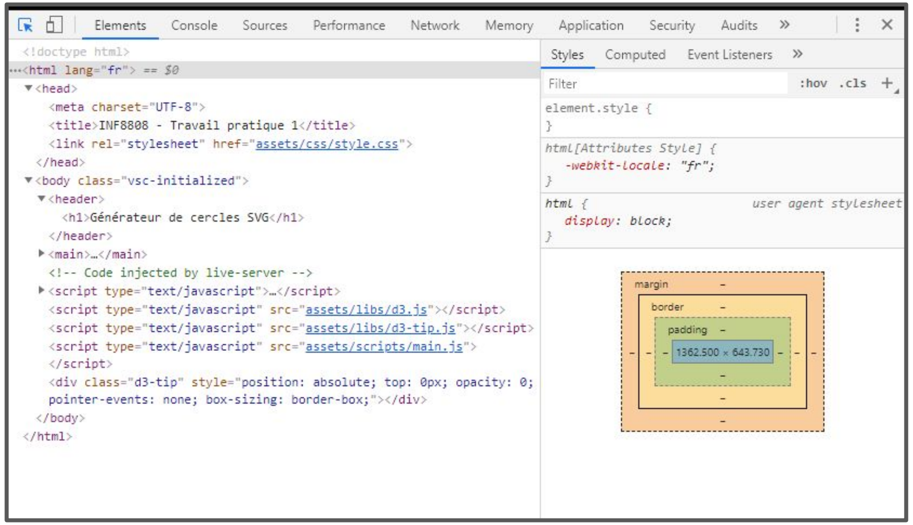
Debugging with Chrome

Debugging with Chrome

You can also use **Firefox** with very similar steps.

1. Right-click anywhere on the page and select “Inspect” <u>OR</u> Ctrl+Shift+I
2. The inspector will open, which is useful for debugging
3. The “Elements”, “Console”, and “Sources” tabs will be the most useful for these TPs (see next slides)

Chrome inspector



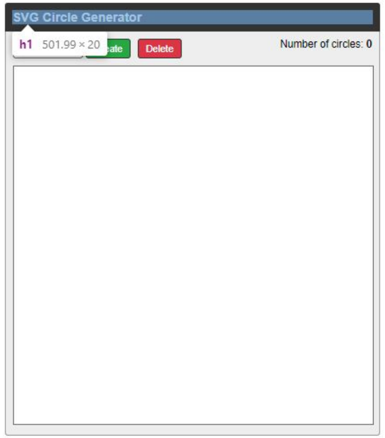
```
<!doctype html>
…<html lang="fr"> == $0
  ▼<head>
      <meta charset="UTF-8">
      <title>INF8808 - Travail pratique 1</title>
      <link rel="stylesheet" href="assets/css/style.css">
    </head>
  ▼<body class="vsc-initialized">
    ▼<header>
        <h1>Générateur de cercles SVG</h1>
      </header>
    ▶<main>…</main>
      <!-- Code injected by live-server -->
    ▶<script type="text/javascript">…</script>
      <script type="text/javascript" src="assets/libs/d3.js"></script>
      <script type="text/javascript" src="assets/libs/d3-tip.js"></script>
      <script type="text/javascript" src="assets/scripts/main.js">
      </script>
      <div class="d3-tip" style="position: absolute; top: 0px; opacity: 0;
      pointer-events: none; box-sizing: border-box;"></div>
    </body>
</html>
```

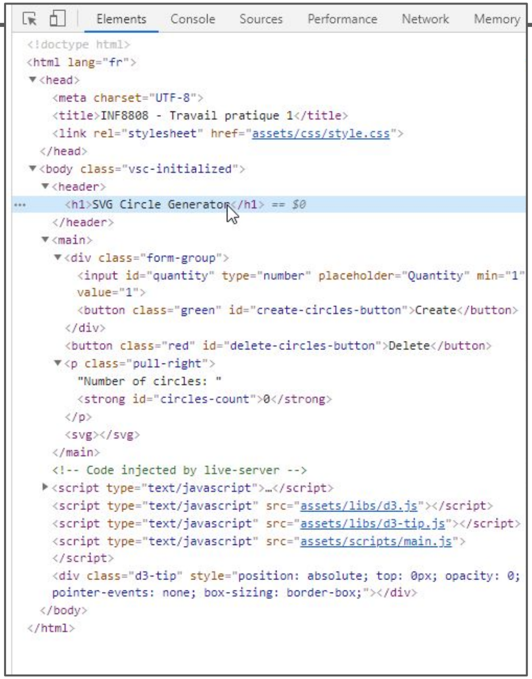
```
element.style {
}
html[Attributes Style] {
   -webkit-locale: "fr";
}
html {                                  user agent stylesheet
   display: block;
}
```

Debugging with Chrome

Element inspection tool

- Shows the HTML structure (DOM) of your code
- Highlights the currently hovered element on the page
- Use it to verify your Python code is correctly generating HTML elements
- Allows you to directly test changes to HTML and CSS
- For more info : [link]





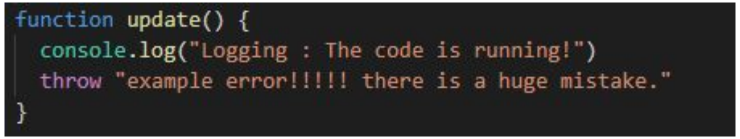
```
<!doctype html>
<html lang="fr">
▼<head>
    <meta charset="UTF-8">
    <title>INF8808 - Travail pratique 1</title>
    <link rel="stylesheet" href="assets/css/style.css">
  </head>
▼<body class="vsc-initialized">
  ▼<header>
...   <h1>SVG Circle Generator</h1> == $0
    </header>
  ▼<main>
    ▼<div class="form-group">
        <input id="quantity" type="number" placeholder="Quantity" min="1"
        value="1">
        <button class="green" id="create-circles-button">Create</button>
      </div>
      <button class="red" id="delete-circles-button">Delete</button>
    ▼<p class="pull-right">
        "Number of circles: "
        <strong id="circles-count">0</strong>
      </p>
      <svg></svg>
    </main>
    <!-- Code injected by live-server -->
  ▶<script type="text/javascript">…</script>
    <script type="text/javascript" src="assets/libs/d3.js"></script>
    <script type="text/javascript" src="assets/libs/d3-tip.js"></script>
    <script type="text/javascript" src="assets/scripts/main.js">
    </script>
    <div class="d3-tip" style="position: absolute; top: 0px; opacity: 0;
    pointer-events: none; box-sizing: border-box;"></div>
  </body>
</html>
```

Inspecting an h1 element

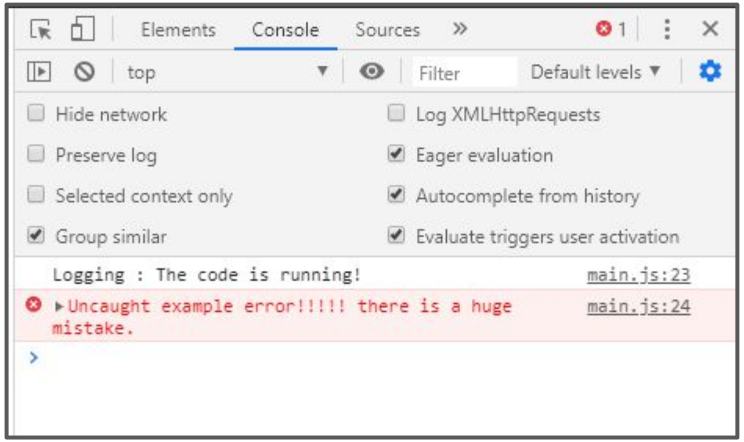
Debugging with Chrome

Console tool

- In the console, you will see outputs from your code
- These may include error messages and logs
- If something is not working, this is the first place you should look
- For more info : [link]



```
function update() {
  console.log("Logging : The code is running!")
  throw "example error!!!!! there is a huge mistake."
}
```



☐ Hide network
☐ Log XMLHttpRequests
☐ Preserve log
☑ Eager evaluation
☐ Selected context only
☑ Autocomplete from history
☑ Group similar
☑ Evaluate triggers user activation

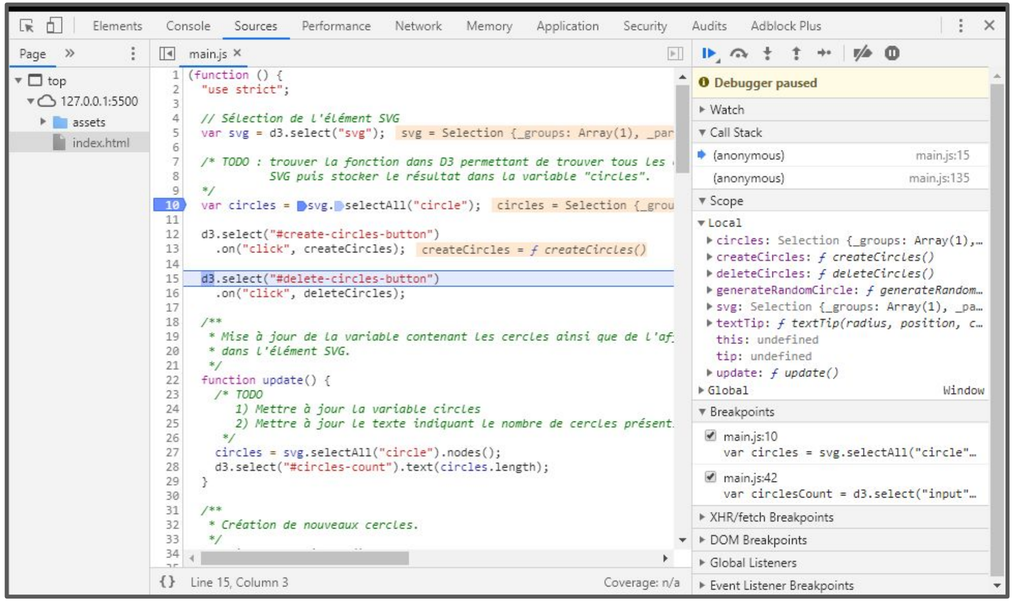
```
Logging : The code is running!                        main.js:23
▶Uncaught example error!!!!! there is a huge          main.js:24
mistake.
```

Logs and errors appear in the console

Debugging with Chrome

Sources inspection tool

- In this tab, you can see your source code and test modification to it
- You can also add breakpoints, where the execution will stop
- From a breakpoint, you can see the value of each variable and step through the code line by line
- For more info : [link]



```
(function () {
  "use strict";

  // Sélection de l'élément SVG
  var svg = d3.select("svg");  svg = Selection {_groups: Array(1), _par

  /* TODO : trouver la fonction dans D3 permettant de trouver tous les
           SVG puis stocker le résultat dans la variable "circles".
  */
  var circles = svg.selectAll("circle");  circles = Selection {_grou

  d3.select("#create-circles-button")
    .on("click", createCircles);  createCircles = ƒ createCircles()

  d3.select("#delete-circles-button")
    .on("click", deleteCircles);

  /**
   * Mise à jour de la variable contenant les cercles ainsi que de l'af
   * dans l'élément SVG.
   */
  function update() {
    /* TODO
       1) Mettre à jour la variable circles
       2) Mettre à jour le texte indiquant le nombre de cercles présent
     */
    circles = svg.selectAll("circle").nodes();
    d3.select("#circles-count").text(circles.length);
  }

  /**
   * Création de nouveaux cercles.
   */
```

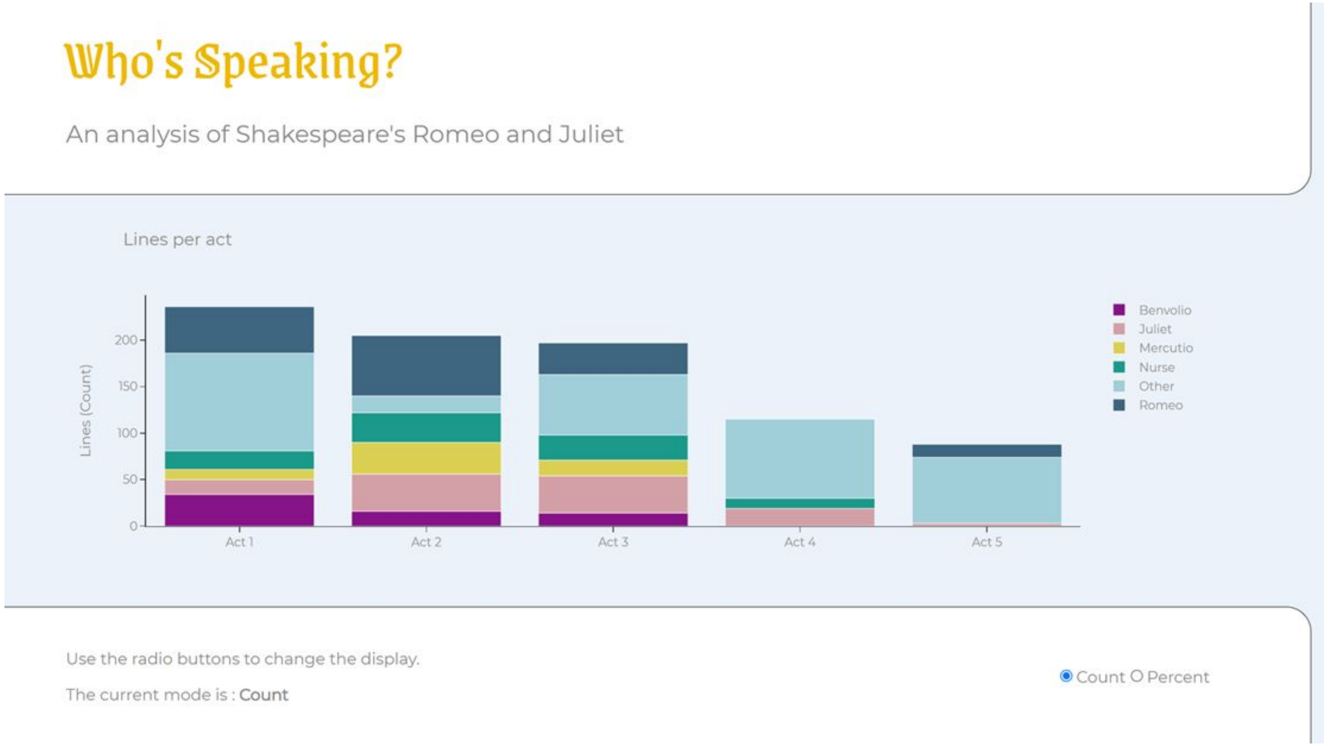
```
Debugger paused
▸ Watch
▾ Call Stack
  (anonymous)                                   main.js:15
  (anonymous)                                   main.js:135
▾ Scope
▾ Local
  ▸ circles: Selection {_groups: Array(1),…
  ▸ createCircles: ƒ createCircles()
  ▸ deleteCircles: ƒ deleteCircles()
  ▸ generateRandomCircle: ƒ generateRandom…
  ▸ svg: Selection {_groups: Array(1), _pa…
  ▸ textTip: ƒ textTip(radius, position, c…
    this: undefined
    tip: undefined
  ▸ update: ƒ update()
▸ Global                                        Window
▾ Breakpoints
  ☑ main.js:10
    var circles = svg.selectAll("circle"…
  ☑ main.js:42
    var circlesCount = d3.select("input"…
▸ XHR/fetch Breakpoints
▸ DOM Breakpoints
▸ Global Listeners
▸ Event Listener Breakpoints
```

Parcours de code

TP2

Introduction au TP2

In this lab, you will implement a stacked bar chart using data from Shakespeare's play, Romeo and Juliet.



Who's Speaking?

An analysis of Shakespeare's Romeo and Juliet

Use the radio buttons to change the display.

The current mode is : Count

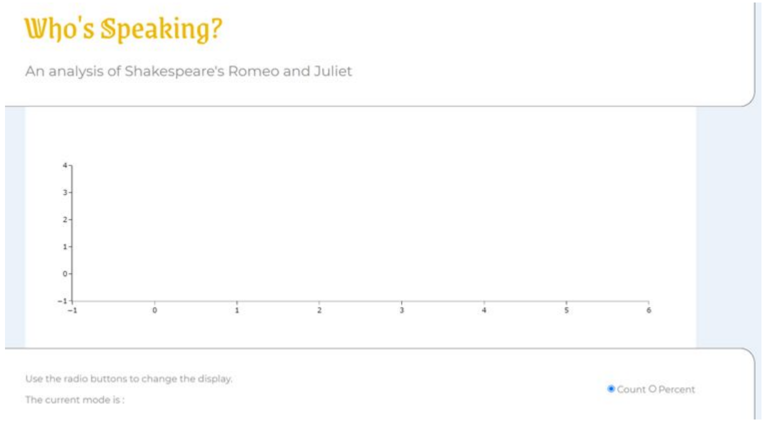
[✓ Count] Percent

Run the code

- In one terminal, at the same directory of *app.py:*

```
python -m virtualenv -p python3.8 venv
venv\Scripts\activate
python -m pip install -r requirements.txt
python server.py
```

- After, open the localhost:8050 in your browser



Who's Speaking?

An analysis of Shakespeare's Romeo and Juliet

Use the radio buttons to change the display.

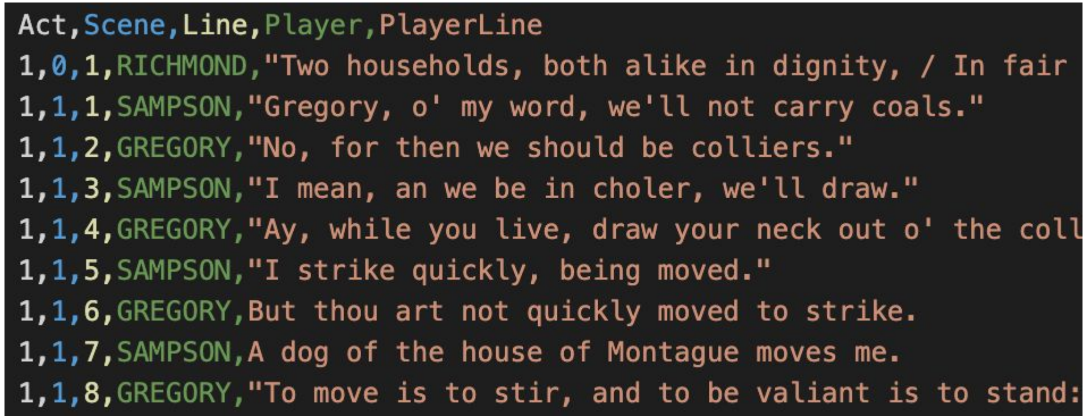
The current mode is :

[✓ Count] Percent

Dataset

The dataset is located in the *src/assets/data/* directory in the archive provided for the lab. The dataset contains the following columns :

- Act : This column represents the act in which the line was uttered.
- Scene : This column represents the scene in which the line was uttered.
- Line : This column tracks the number of the line for the given act and scene.
- Player : This column contains the name of the player who uttered the line.
- PlayerLine : This column contains the content uttered by the given player during the given act and scene at the line with the given number.



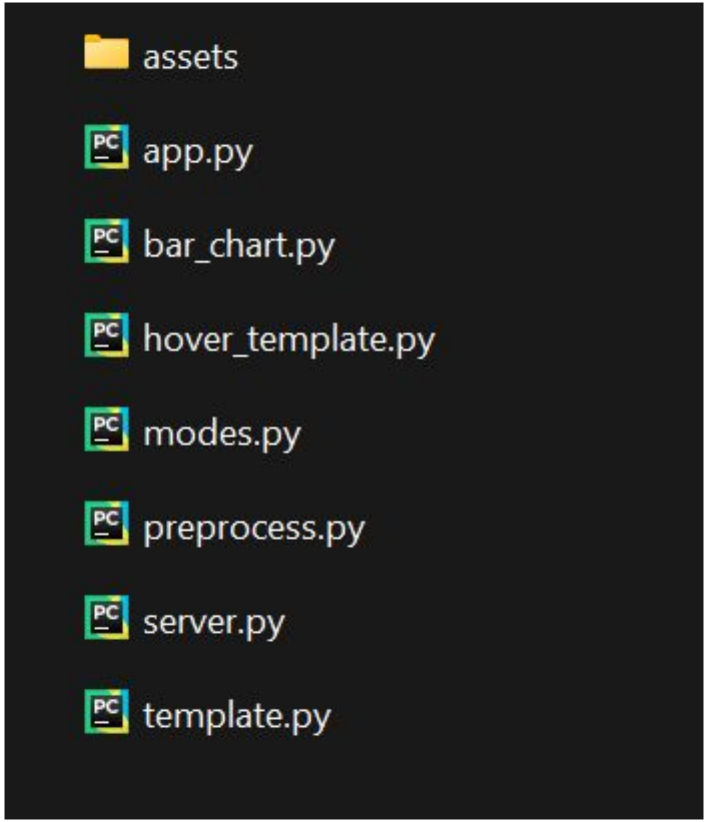
```
Act,Scene,Line,Player,PlayerLine
1,0,1,RICHMOND,"Two households, both alike in dignity, / In fair
1,1,1,SAMPSON,"Gregory, o' my word, we'll not carry coals."
1,1,2,GREGORY,"No, for then we should be colliers."
1,1,3,SAMPSON,"I mean, an we be in choler, we'll draw."
1,1,4,GREGORY,"Ay, while you live, draw your neck out o' the coll
1,1,5,SAMPSON,"I strike quickly, being moved."
1,1,6,GREGORY,But thou art not quickly moved to strike.
1,1,7,SAMPSON,A dog of the house of Montague moves me.
1,1,8,GREGORY,"To move is to stir, and to be valiant is to stand:
```

Tasks to be accomplished :

1. **Data pre-processing**
   - File : ./src/*preprocess.py*
2. **Creation of the bar chart**
   - File : ./src/bar_chart*.py*
3. **Completing a template**
   - File : ./src/*template.py*
4. **Adding a tooltip**
   - File : ./src/*hover_template.py*
5. **Display mode toggle**

   - File : ./src/hover_app.py

<u>The other files should not be modified.</u>

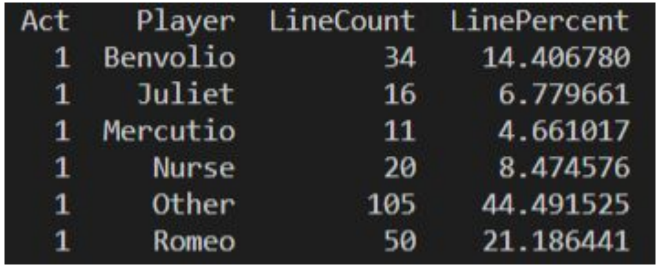

assets
app.py
bar_chart.py
hover_template.py
modes.py
preprocess.py
server.py
template.py

1. Data preprocessing

You will need to fill the 3 function on the file **preprocess.py** :

1. ***Summarize_lines*** : For each act, group each player's lines together, summing each player's line count for the act and including the corresponding percentage of lines for that player in that act
2. ***Replace_others*** : Modify the structure to include an "Other" category, which contains the sum of the count and percentage of lines in that act of players not belonging to the top 5 for the play
3. ***Clean_names*** : Update the names of the players in the dataset so they follow correct capitalization

The result will generate this type of structure:

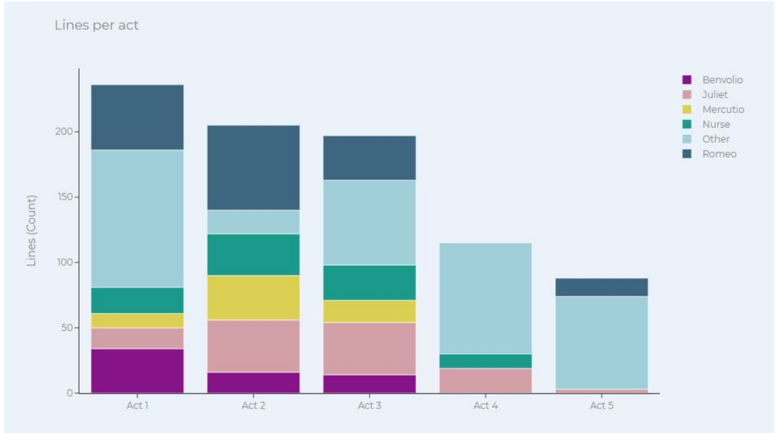


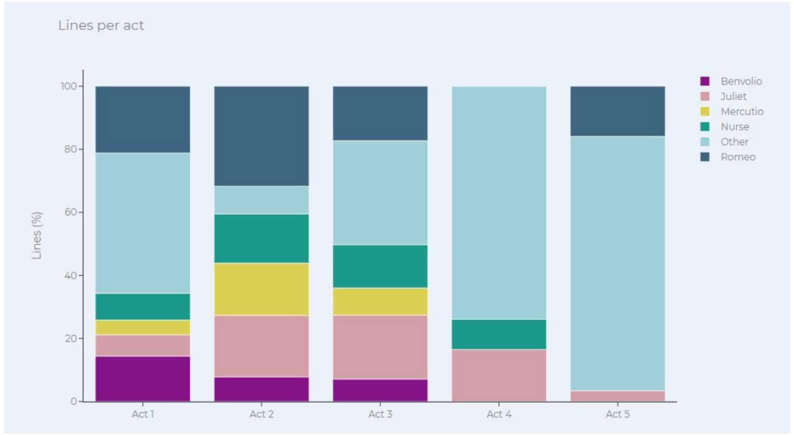
```
Act     Player  LineCount  LinePercent
  1   Benvolio         34    14.406780
  1     Juliet         16     6.779661
  1   Mercutio         11     4.661017
  1      Nurse         20     8.474576
  1      Other        105    44.491525
  1      Romeo         50    21.186441
```

2. Bar Chart

You will need to fill the 3 function on the file **bar_chart.py** :

1. ***Init_figure*** : Complete the definition of the chart's figure to include the template and title to be used,
2. ***Draw*** : Display the appropriate data in the bar chart depending on the current mode,
3. ***Update_y_axis*** : Display the appropriate text on the y-axis depending on the current mode





3. Template

The code for this part is in the file ***template.py***.

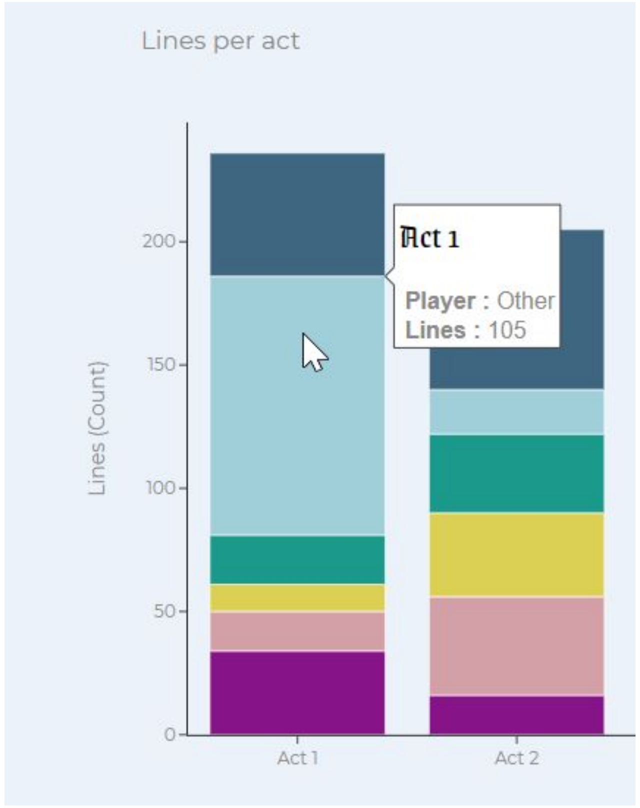
You need to complete the function ***create_template***. Make sure to closely follow the instructions in the comments of the code when defining each element of the theme. The desired values to use are defined in a variable at the beginning of the file. Make sure the colors in the bar chart appear in the same order as in Figures

4. Tooltips

The tooltip should contain as a title the act associated to the bar's group, as well as the bar's player and line count or percentage (depending on the current mode).

The comments in the code give a detailed description of the contents and appearance of the tooltip. See Figure on the side for the expected final result.

The code for this part should be written in the file ***hover_template.py*** in the function ***get_hover_template***

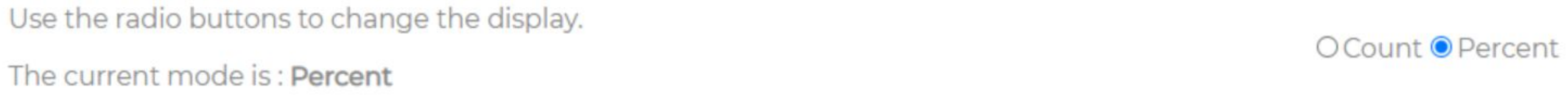


5. Display mode toggle

You will implement a method to toggle between the chart's to display modes based on the radio button input. Whenever the input on the radio button is modified, the bar chart should be modified correspondingly.

The displayed data should be selected from the appropriate column in the dataset depending on the selected mode. The tooltip and y-axis should also display texts corresponding to the selected mode. Further, the information text at the left of the footer should contain the currently selected mode.

To complete this part, you will need to find the appropriate functions to call in the ***radio_updated*** function in ***app.py***. Beginning by partially completing this function may also be helpful for visualizing your bar chart while you complete the previous steps.


Use the radio buttons to change the display.

The current mode is : Percent

Count [✓ Percent]

Some tips to get started

Data preprocessing

- Avoid direct manipulation of data indices where possible (i.e. a loop for index i → data [i])
  - This practice will improve the quality and maintainability of the code while reducing the risk of errors
- Instead, explore Pandas' DataFrame methods, such as:
  - `.groupby()`
  - `.concat()`
  - `.replace()`
  - `.sort_values()`
  - `.sum()`
- Using `for ... of` is also sometimes useful

Overall quality and clarity of the submission

More details

Each TP will also be graded on the overall quality and clarity of the submission .

Examples

- Clear code structure
- Do not modify signatures of existing functions
- You can add new functions, but they must be clear and their addition must be justified
- Use clear indentations
- Add comments as needed, but not too many
- Don’t leave dead code
- Don’t leave useless console.log
- Make sure to follow instructions for submission

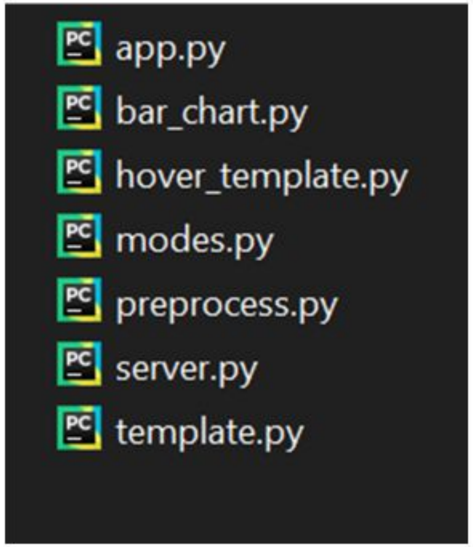
Etc.

All this points can affect your grade on the Overall Quality score

Submission

More details about the submission

- You must submit a .zip file (not .rar) containing the files you received on code.zip **OR** just the files listed on the image.
- Don’t send the ***venv*** folder in your zip or any other file that you haven’t received or was asked for.
- You will upload your file on the Python box available on Moodle.


app.py
bar_chart.py
hover_template.py
modes.py
preprocess.py
server.py
template.py