FONDAMENTAUX

INF1013, Hiver 2021

DMI, UQTR

RAPPELS HTML, JS, DOM

- Langage de balisage, rendu par un navigateur.
- Les Balises englobent les éléments.

  <**h1**> Grand Titre</**h1**>
- Les pages sont organisées selon le modèle DOM

  window->document->html->{head, body->balises}
- Le JS est un langage de programmation.
- Le JS est interprété par le navigateur.
- Le JS manipule le HTML en se le représentant selon le DOM

RAPPELS HTML, JS, DOM

- Balises HTML déterminent des éléments sémantiques du documents.
- Les attributs (souvent confondus avec les propriétés) placés dans la balise ouvrante apportent de la précision à la balise.

```
<h1  enabled="true">Grand Titre</h1>
```

- Le JS peut manipuler ces attributs.

```
document.querySelector("h1").setAttribute('enabled', 'false');
```

- Il est permis aussi d'étendre les attributs en ajoutant ses propres attributs non natifs du standard w3.

```
<h1  uqtr ="allo !"> Grand Titre</h1>
```

- JS pourra cibler et lire le contenu de cet attributs comme tout autre attribut natif.

RAPPELS HTML, JS, DOM

- Il existe une différence importante entre Attribut et Propriété en HTML.
  - Les deux notions ont les mêmes nom Ex. *id, class, enabled, etc*.
  - Les attributs sont définis et initialisés par le HTML
  - Les propriétés sont accédés à partir des nœuds du DOM
- Exemple: Quand le navigateur rend

  `<input type="text" value="Sarah">`
  - Il crée un nœud correspondant dans le DOM avec une propriété value initialisé avec le texte "Sarah".
  - Si l'utilisateur saisit "Michelle", qu'obtient-on avec le code JS suivant?

RAPPELS HTML, JS, DOM

```
attr = document.querySelector("input").getAttribute('value');
prop = document.querySelector("input").value;
```

- **attr** restera inchangé et gardera donc la valeur "Sarah"
- **prop** prendra la valeur "Michelle".

- Remarque fondamentale:
  - L’ attribut spécifie la valeur initiale de ***value***
  - La propriété donne la valeur courante de ***value***.

ANGULAR

INF1013, Hiver 2021

DMI, UQTR

INSTALLATION

- Angular requiert l'installation de NodeJs (une technologie JS serveur).
  - https://nodejs.org/en/
- **npm** est le gestionnaire de paquettages de NodeJs.
- Nous utiliserons ses commandes pour installer Angular
  - *ng --version* permet d'avoir la version de Node installée.
  - Elle doit être > 10.9.0 pour installer Angular.
- Nous utiliserons Angular à l'aide du CLI
  - interface en ligne de commande
  - *npm install -g @angular/cli* va installer le CLI de Angular
  - Une fois le CLI installé, *ng new mon-app* permettra de créer un projet Angular
  - Dans le dossier mon-app, la commande *ng serve –open* exécutée lance l'application.

LE MODEL MVC

- Angular utilise le modèle MVC ou ses dérivées MVP pour gérer les documents HTML.
- Dans ce modèle,
  - Les vues (Views) sont les documents HTML appelée aussi Template.
  - Les contrôleurs (Controllers) sont des fichiers source contenant des classes JavaScript (JS) ou TypeScript (TS).
  - Les modèles sont les données transitant entre les Views et les Controllers.
- Nous utiliserons TypeScript (TS) à la place de JavaScript pour son typage faible.
  - https://www.typescriptlang.org/docs/handbook/declaration-files/by-example.html

COMPOSANTS ET MVC.

- Tout triplet Model, View, Component est appelé composant et peut définir une nouvelle balise personnalisée appelée sélecteur.
- Ce sélecteur peut ensuite être utilisé dans la View d'un autre composants.
- Ainsi sera construite par composition, l'interface web complète.

```
import { Component } from '@angular/core';

@Component({
        selector: 'app-root',
        templateUrl: './app.component.html',
        styleUrls: ['./app.component.css']
})
export class AppComponent {
         title = 'inf1013';
         entry = 'Michel';
}
```

COMPOSANTS ET MVC

- Il existe dans la racine du projet, une page index.html (racine) contenant le code des balises html, head et body.
- Il est dès lors inutile de les déclarer dans la view d'un composant.
  - Les composants constituent une ou plusieurs parties de la pages racine.
- Pour lier les données entre la View (Template) et le Contrôleur, Angular utilise (dans les templates)
  - L'interpolation {{}}
  - Le binding [()]

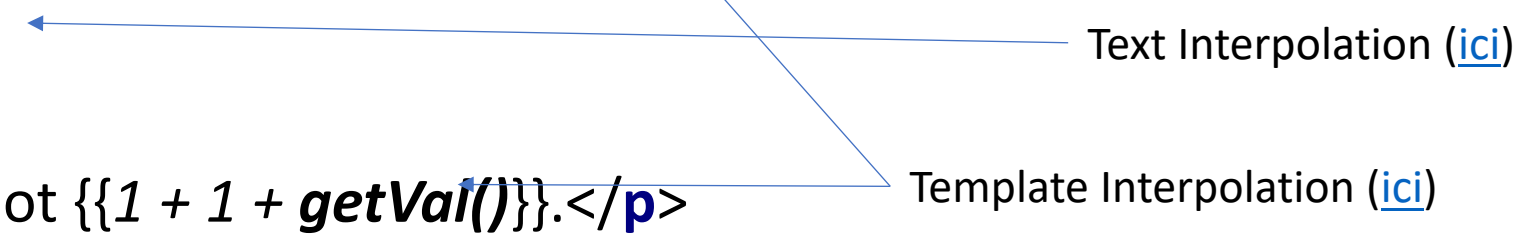
INTERPOLATION

- L'interpolation permet d'incorporer des valeurs calculées / issues de variables membres ou de templates dans
  - Les textes d'éléments HTML : On parle de template interpolation
  - Les propriétés d'éléments HTML On parle de text interpolation
  - Caractérisée par {{ expression}}
- Les expressions du modèle permettent de les calculer
- Exemple.

```
<h1>La variable titre du contrôleur contient : {{title}}</h1>
<img src = {{iconUrl}}/>
```

- Opérations simples

```
<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}.</p>
```



Binding et Statement Expression

- Pour assigner les expressions aux propriétés des balises (nœuds du DOM), l'interpolation textuelle **** est à éviter.
- Angular préfère le binding avec l'opérateur […].

  ****
  - Cet opérateur indique que les données passent de la variable ***iconUrl*** vers l'attribut ***src*** de la balise image.
  - Quant un attribut est entre **[]**, sa propriété reçoit les donnés résultant de l'expression à droite.
- Pour lier un évènement à une fonction, l'opérateur **(…)** indique que l'information passe du Template vers le contrôleur.

  <button (click)="displayMsg()">Afficher</**button**>

Binding: résumé

Type	Syntaxe	Catégorie
Interpolation Property Attribute Class Style	{{expression}} [target]="expression" bind-target="expression"	One-way du contrôleur à la vue
Event	(target)="statement" on-target="statement"	One-way de la vue au contrôleur
Two-way	[(target)]="expression" bindon-target="expression"	Two-way

DIRECTIVES

- Les directives sont des composants de type **attribut html**. propriété du DOM
- Angular prédéfinit plusieurs directives.
- Il est possible de définir des directives personnalisées

<balise directive ></balise>

Directives Structurantes: Aperçu

- Les directives structurantes permettent de modifier le DOM. Elles commencent par *
- Il existe plusieurs directives prédéfinies. (***ngFor, *ngIf etc.**)
- Elles peuvent être liées avec des expressions.

   <li *ngFor="let s of students">{{s}}
- Où la variable *students* est définie dans le contrôleur comme:
  - **students**: **string**[] = ['Jean', 'Jacques', 'Isabelle', 'Andrée’];
- Exemple avec la directive *ngIf:
  - <p *ngIf="students.length <= 0">Liste vidée</p>

Directives Non-Structurantes: Aperçu

- Il existe une autre famille de directives issue du core de Anglular qui ne modifient pas la structure du DOM.
- Exemple :
  - ngClass

    Redéfinmition de l'attribut classe

    ```
    <some-element [ngClass]="'first second'">...</some-element>
    ```
  - ngStyle:

    ```
    <some-element [ngStyle]="{'max-width.px': widthExp}">...</some-element>
    ```

INPUTS

- Les inputs permettent de passer les informations entre composants.
- Les inputs sont utilisés comme des attributs pour les nouvelles balises-composants.

Entrées d’un Composant

- Il est possible de créer plusieurs composants avec Angular comme statué plus tôt, avec le CLI
  - ***ng generate component*** <nom-du-composant>
- La commande génère un nouveau composant.
- Ce composant peut alors être utilisé, grâce à son **selecteur** comme composant dans le template d’un autre composant (parent).
- Pour passer des informations d’un composant vers son composite (de Parent à enfant), nous utilisons le décorateur **@Input(**)

PASSAGE D'INFORMATION PARENT-ENFANT

- DANS LE COMPOSANT PARENT
  - Dans le template, on pourra lier une expression comme suite
  - <**selecteur-enfant [mesgParent] = "Coucou du parent !"**></**selecteur-enfant**>

- DANS LE COMPOSANT ENFANT
  - On définit
  - @Input() **mesgParent**: string; comme une variable membre du contrôleur.
  - On pourra utiliser cette variable membre spéciale dans le contrôleur et le template.

Exercice d'Application: Étape 1

- L'objectif de cet exercice est de réaliser une SPA (Single Page Application) qui gère une liste d'étudiants
  - Créer un nouveau projet avec Angular CLI nommé inf1035
    - Créer un dossier models contenant la classe **Student** avec les champs
      - fname**:string**, lname**:string**, cp**:string**, et scores: **Score[]**
    - Créer la classe **Score** avec les champs
      - name:**string** (examen, travail de groupe et value:**string** (A, A+….)
  - Dans le composant principal (app-root)
    - Ajouter dans le contrôleur une variable membre students: **Student**[]
    - Pré-remplire (données mock) la liste d'étudiants dans le constructeur du contrôleur.
  - Dans le template du composant app-root
    - Afficher la liste/tableau des étudiants (cp, nom, prenom) à l'aide de la directive *ngFor

EA: Étape 2, L'écoute des évènements

- Dans le contrôleur du composant app-root:
  - Ajoutez la variable membre selected:Student
  - Ajoutez une méthode select(ss:Student)
    - Cette méthode initialisera la variable *selected* par ss.
- Dans le template du composant app-root:
  - invoquez la méthode select(…) lors du click sur un item de la liste des étudiants
  - Utiliser (click)

Étape 3: Composant *Details*.

- Créez le composant ***Details***
  - Déclarez une variable membre student:Student.
  - Dans le template de détails affichez les détails des informations de l'étudiant.
    - Formatez-les correctement à l'aide des CSS.
  - Décorer cette variable membre avec @input pour passer l'instance de *Student* au composant ***Details***.
  - A l'aide de son selecteur, ajoutez le composant ***Details*** dans le template du app-root.
  - Utilisez la directive ngClass pour mettre les notes < 90 en rouge et >= 90 en bleu