

FONDAMENTAUX

INF1013, Hiver 2022

DMI, UQTR

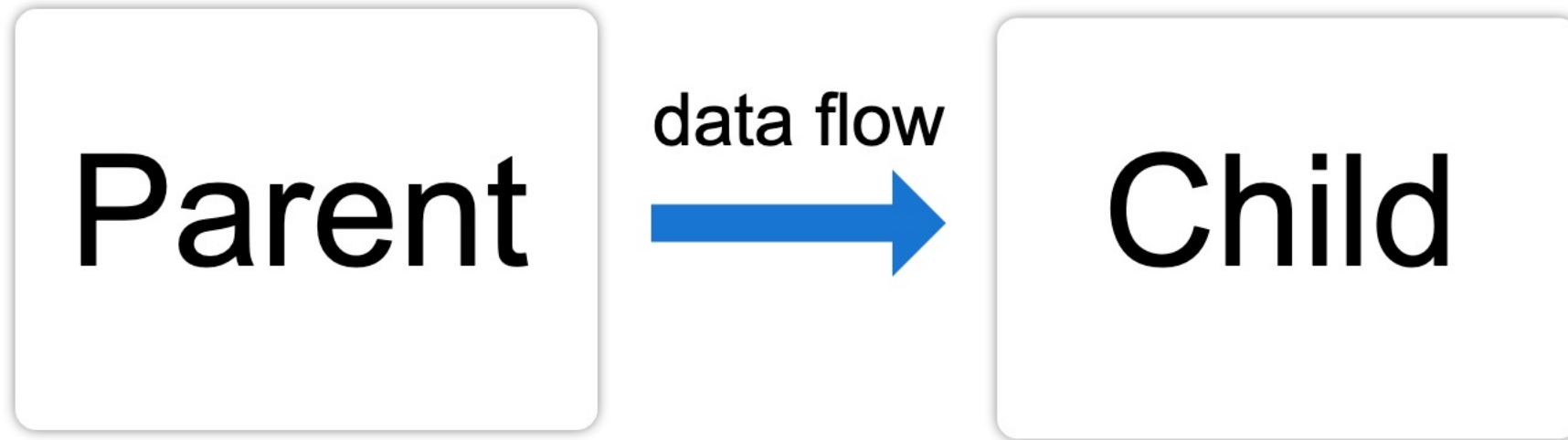
Entrée: @Input

Entrée des Composants

- Problématique: On veut envoyer des informations d'un composant à son composite. (composant enfant).
 - **On veut afficher les détails d'un étudiant de la liste contenu dans le composant parent.**
- Le composant parent ici est le *app.component* qui contient la liste.
- Le composant enfant est ici *details.component*.
- Le parent contient le sélecteur de l'enfant dans son template.
 - Ce sélecteur sera le slot où s'insérera le composant enfant *details.component*.

Entrée des Composants

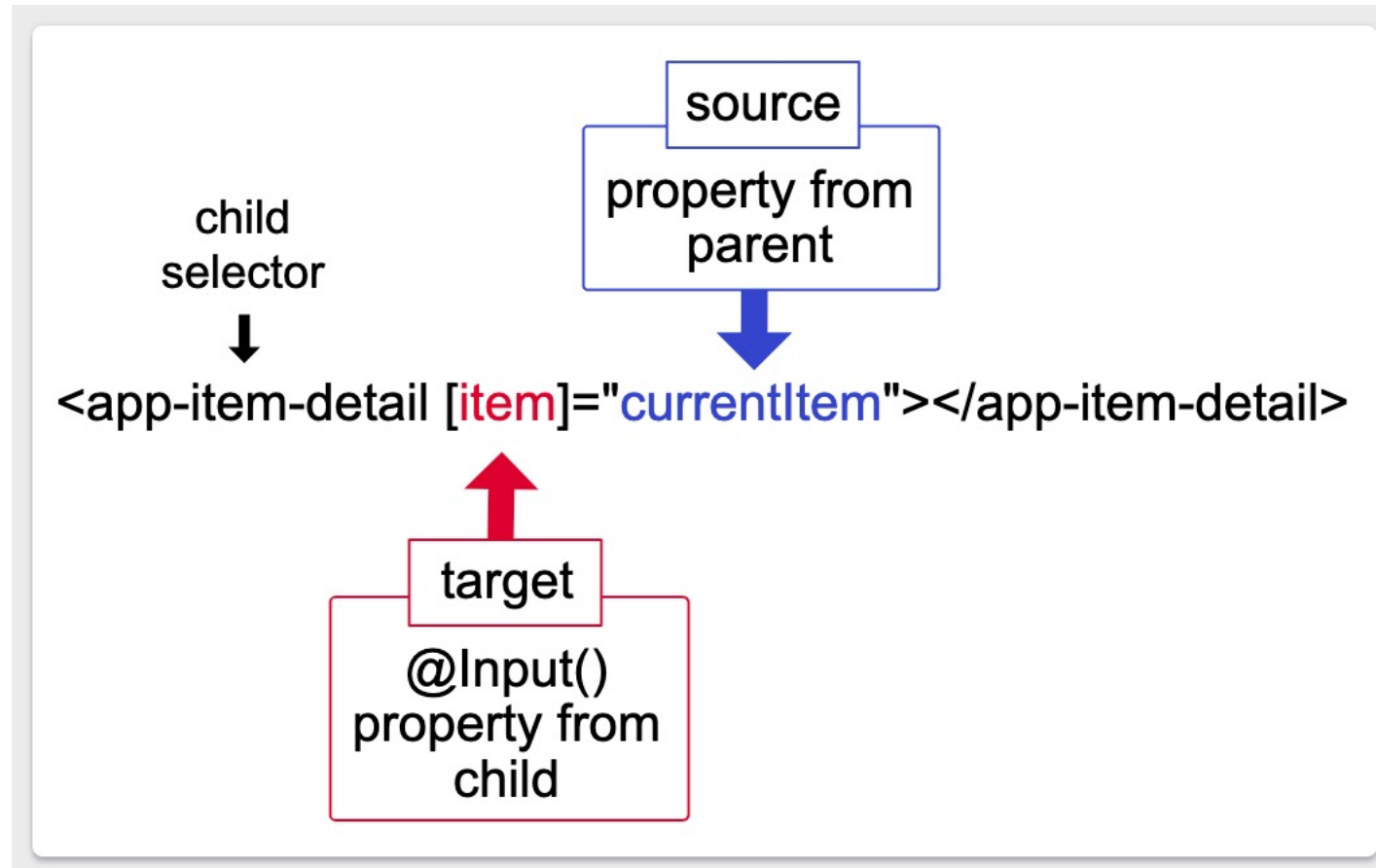
 @Input



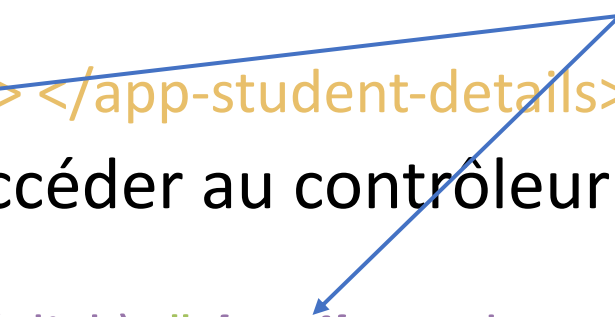
Entrées d'un Composant

- Angular utilise les *entrées* pour résoudre ce problème.
- Les entrées sont des variables membres annotées par **@Input()** dans le composant enfant.
- Pour passer des informations d'un composant vers son composite (Parent à enfant), nous utilisons un binding (UI -> Contrôleur) **[]** dans le template du parent.
 - **@Input()** **msgParent**: string;
 - **msgParent** est une variable membre du contrôleur enfant.
 - Le template du parent pourra alors lier une expression (statement expression)
 - **<sel-enfant [msgParent] = "Coucou du parent !!!!!"></sel-enfant>**

Entrées d'un Composant: Template Parent



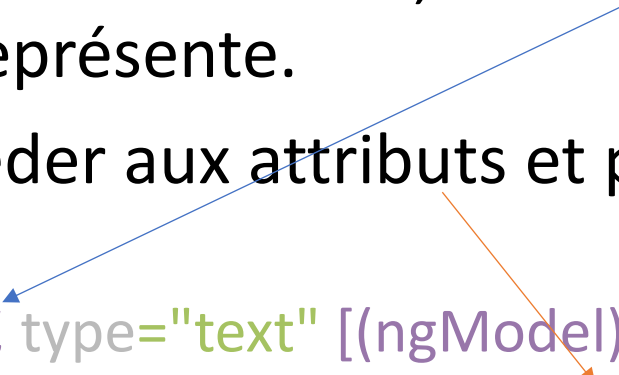
Alternative Parent-enfant : Variable de Template

- Les templates var (Tvar) sont des variables déclarées dans le template sans lien avec le contrôleur.
 - Ils se déclarent à l'aide de # suivi du nom de la variable.
 - Ce sont des directives (attribut) en réalité
 - Exemple <Comp **#var1**></Comp>
- Lorsque une Tvar est déclarée dans un composant, elle va référencer son contrôleur
`<app-student-details #details></app-student-details>`


Le contrôleur de app-student-details
- Il devient alors possible d'accéder au contrôleur et aux vm de l'enfant dans le template parent
`<li *ngFor="let s of students" (click)="details.student = s">`

Variable de Template

- Plus largement, il est possible de déclarer une Tvar dans n'importe quelle balise de template.
- Déclarée dans une balise standard, une tvar va référencer l'objet du DOM que la balise représente.
- On pourra alors accéder aux attributs et propriétés des balises.
 - Exemple



```
<input #ref1 type="text" [(ngModel)]="model" />
<span *ngIf="true">Value: {{ ref1.value }}</span>
```

À lire: la portée des tvars: <https://angular.io/guide/template-reference-variables>

Alternative Parent-enfant : @ViewChild

- Il existe une autre manière d'accéder à l'instance du composant détail.
- Cette approche utilise l'annotation @ViewChild

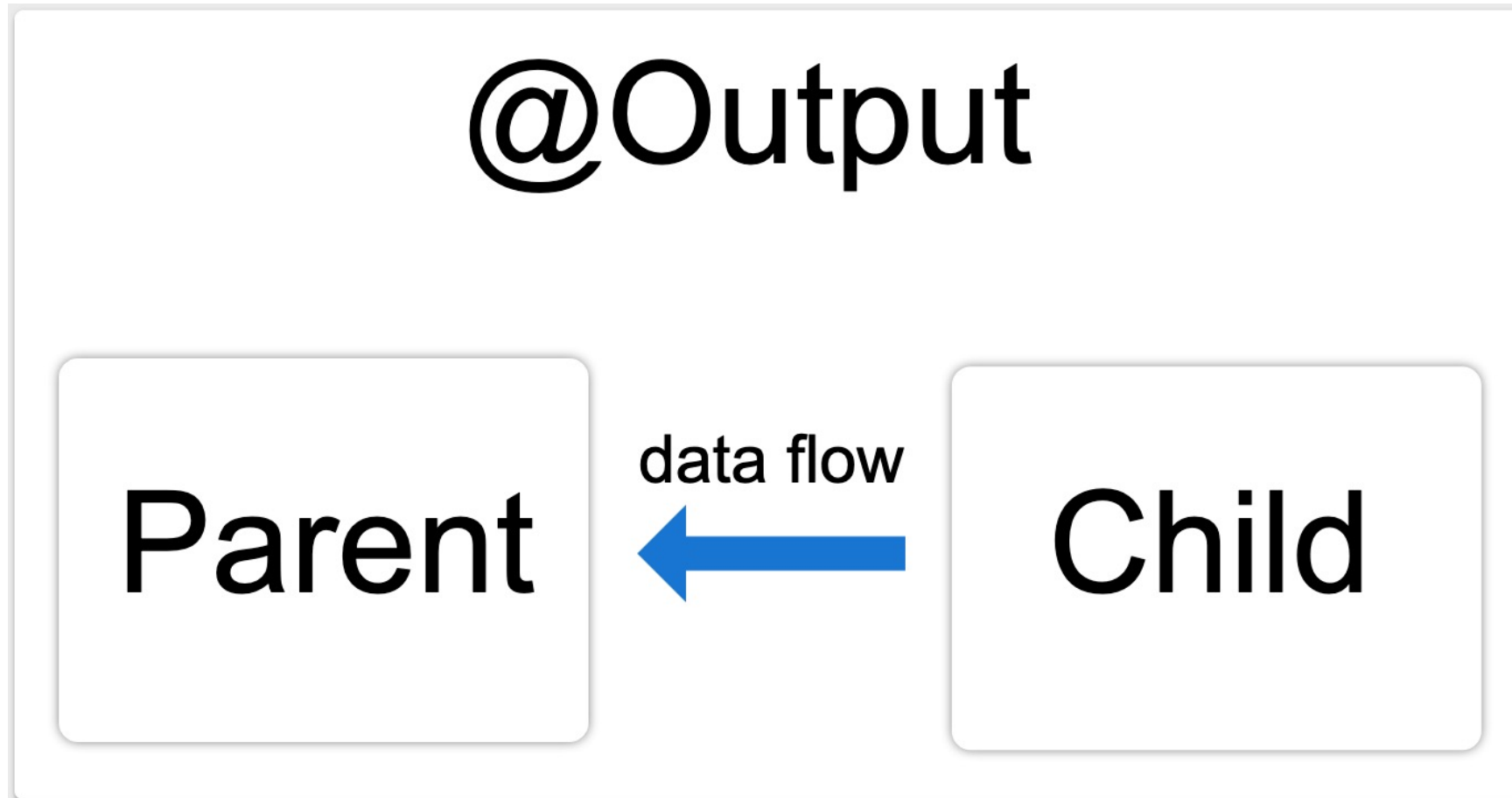
```
@ViewChild(DetailsComponent, {static: true})
detailsComponent: DetailsComponent;
```
- Cette déclaration se fait dans le contrôleur parent.
 - Le premier paramètre du décorateur désigne le composant.
 - Le deuxième désigne le type de résolution
 - (avant ou après la détection de changement)

Exercice d'Application TVar et @ViewChild

- Reprendre le Lab précédent en utilisant une template variable à la place de @Input.
- Refaire l'exercice avec cette fois-ci, l'utilisation du décorateur `@ViewChild()`

Sortie: @Output et EventEmitter

Sortie des Composants



Exercice d'application: @Output

- Partir du Lab précédent.
 - Rajouter les boutons de navigation < et > dans *details*.
 - A l'aide @Output notifier le parent en change l'item actif.

	MENDEL20202311	Jean-Paul MENDEL	B
	PELER2021021	André PELLERIN	B
	OUELLET20120104	Bertrand OUELLETTE	C
	MARCHAN199504101	Giles MARCHAND	C
	Gendr198511231	Stéphanie GENDRON	C
	MARINJ198412124	Julie MARIN	C+
	Croteau200504314	Guylaine CROTEAU	B-

Prénom: André
Nom: Pellerin
Sexe: Masculin
Code Permanent: PELER2021021
Moyenne Globale: 83.6, **B**

#	Dates	Évaluations	Notes
0	12/03/2021	TP I	84/100
1	16/04/2021	TP II	70/100
2	22/05/2021	TP III	93/100
3	19/04/2021	Examen I	95/100
4	09/05/2021	Examen II	76/100

<

>

Sortie des Composants

- Problématique: On veut envoyer des informations d'un composant enfant à son composeur. (composant parent).
 - **On veut faire de la « navigation » dans details.**
 - **Les boutons de précédent (<) et suivant (>) de navigation sont dans details.**
- Ici, le composant parent est *app.component* qui contient la liste.
- Le composant enfant est *details.component* avec ses boutons de nav
- Le parent contient le sélecteur de l'enfant dans son template.
 - Ce sélecteur sera le slot où s'insérera le composant enfant *editor.component*.
 - L'envoi d'infos du parent vers l'enfant se fera par la technique des entrées (@Input) vues précédemment.

Sortie d'un Composant et Évènement

- À l'inverse des entrées, il est possible qu'un composant enfant veuille envoyer des informations à son parent. On dit que ce composant a des sorties.
 - Les sorties de l'enfant vers le parent utilisent des émetteurs d'évènements.
- Le décorateur **@Output** est alors utilisé en combinaison avec **EventEmitter** dans le composant enfant.

```
@Output() navRequest = new EventEmitter<number>();
```

-> Le typage de EventEmitter détermine le type de donnée envoyée au destinataire.

- Dans le template parent, on retrouve un binding dans le sens UI -> Contrôleur: ().

```
<app-student-details class="flex2"
  *ngIf="index >= 0"
  [student] = students[index]
  (navRequest)="next($event)">

</app-student-details>
```

Sortie d'un Composant et Évènement

- **\$event** est un mot-clé qui contiendra l'argument de l'évènement (ici: number)
- Ce binding dit : Si l'émetteur **navRequest** émet un évènement, alors appelle la méthode **next(inc: number)** dans le contrôleur parent.

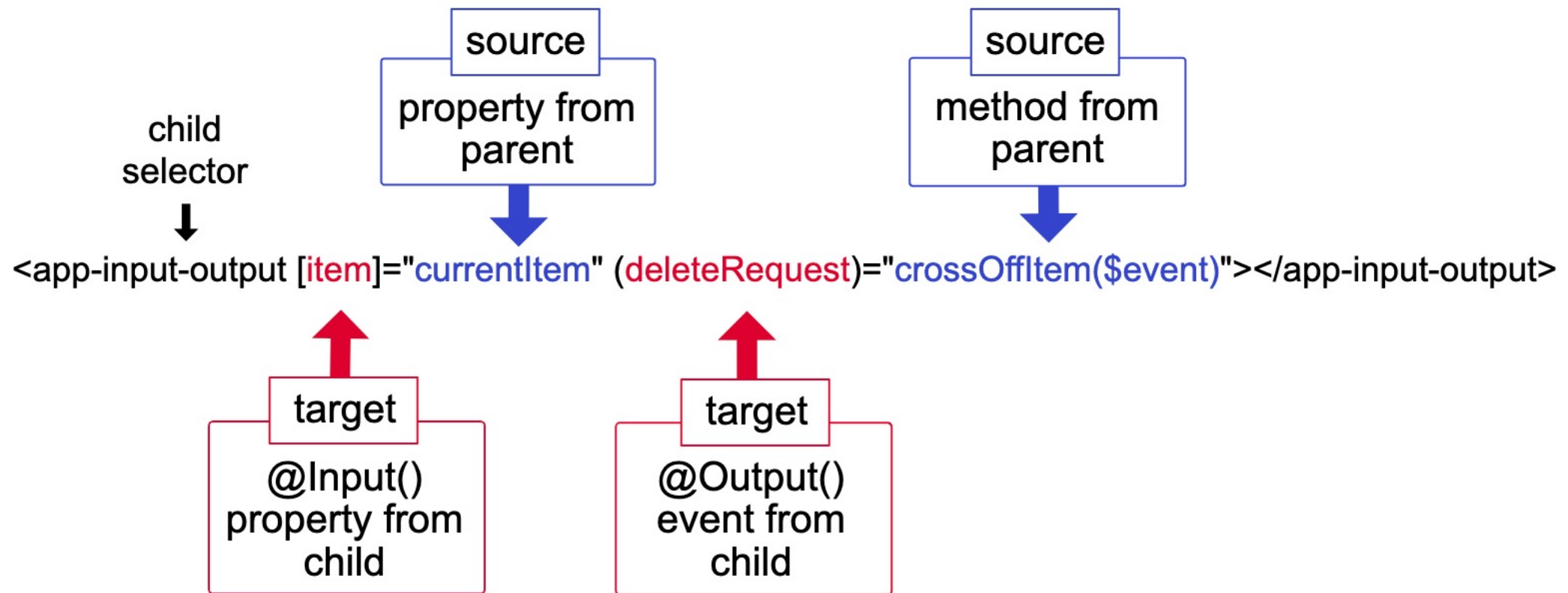
```
next(inc: number) {  
    this.index = (this.index + inc) % this.students.length;  
}
```

- Le changement de la variable membre index va automatiquement mettre à jour le UI
- Dans le template **details.component**, on réagit aux boutons < et > en émettant l'évènement **navRequest.emit(+/- 1)**

```
<nav>  
  <button (click)="navRequest.emit(-1)"></button>  
  <button (click)="navRequest.emit(1)"></button>  
</nav>
```


Sortie et Entrée

Sortie et Entrée séparée d'un Composant



Entrée-Sortie 2-way binding.

- Pour que la liaison de données bidirectionnelle fonctionne, la propriété `@Output()` doit utiliser le modèle d'écriture ***inputChange***, où ***input*** est le nom de la propriété `@Input()`.
- En d'autres termes:
 - Si la propriété `@Input()` est nommée ***size***,
 - Alors la propriété `@Output()` doit être nommée ***sizeChange***.

- Exemple

```
@Input() size: number;  
@Output() sizeChange = new EventEmitter<number>();
```

- Alors seulement on pourra écrire le « banana in the box » `[(())]`

```
<app-student-details [(size)]="fontSizePx"> </app-student-details>
```

Les Pipes

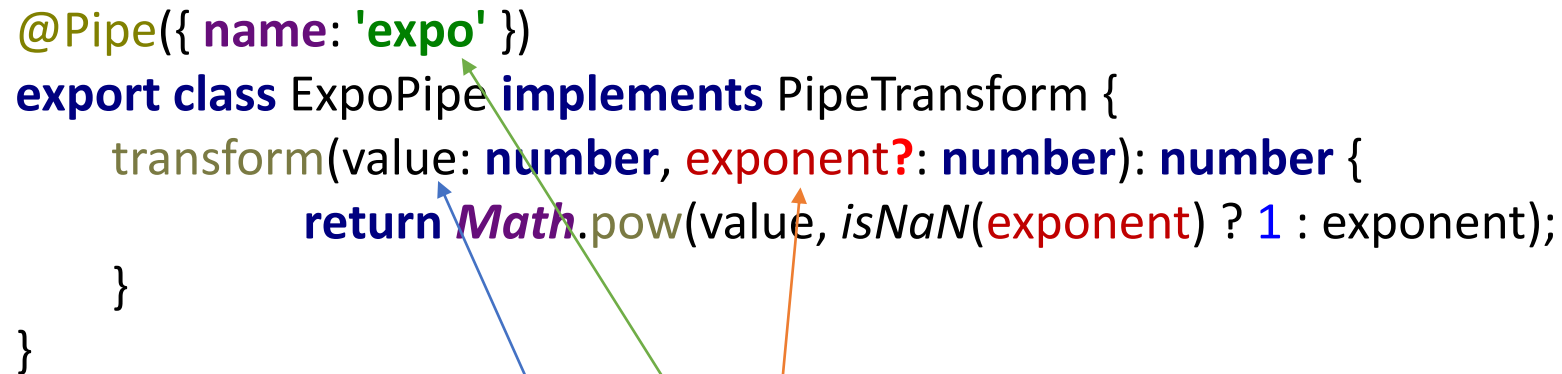
Pipes (Composants de chaînage)

- Ce sont des composants qui raccordent les sorties (dans une expression d'interpolation) vers leurs entrées dans le but des les transformer.
 - Ils permettent par exemple de formater des sorties.
- Angular contient plusieurs pipes prédéfinis.
 - *Exemple date, json, uppercase...*
- Ainsi s'utilisent les pipes dans le template
`<p>{{student.lname | uppercase}} </p>`
- Certains pipes prennent des paramètres
`<p>Date de naissance: {{student.dob | date : "dd/MM/yy"}}</p>`
- Il est possible alors de chaîner les pipes

Composants Pipes (Custom)

- On peut créer ses propres pipes (personnalisés) avec Angular.
 - Avec la commande **ng generate pipe** <nom-du-pipe>
- Exemple
 - Nous voulons créer un pipe 'Expo' calculant l'exposant d'un nombre dans une expression.








```
@Pipe({ name: 'expo' })
export class ExpoPipe implements PipeTransform {
  transform(value: number, exponent?: number): number {
    return Math.pow(value, isNaN(exponent) ? 1 : exponent);
  }
}
```



- Utilisation: `<p>{{ 12 | expo:2 }}</p>`

Exercice d'application: Pipe

- Créer un pipe convertissant les notes des étudiants en cote.
 - Utilisez les intervalles standard du plan de cours.
- Afficher les cote des étudiants à côté de leurs notes
 - Dans les composants app
 - Dans le composant details

	MENDEL20202311	Jean-Paul MENDEL	B
	PELER2021021	André PELLERIN	B
	OUELLET20120104	Bertrand OUELLETTE	C
	MARCHAN199504101	Giles MARCHAND	C
	Gendr198511231	Stéphanie GENDRON	C
	MARINJ198412124	Julie MARIN	C+
	Croteau200504314	Guylaine CROTEAU	B-

Prénom: André
Nom: Pellerin
Sexe: Masculin
Code Permanent: PELER2021021
Moyenne Globale: 83.6, B

#	Dates	Évaluations	Notes
0	12/03/2021	TP I	84/100
1	16/04/2021	TP II	70/100
2	22/05/2021	TP III	93/100
3	19/04/2021	Examen I	95/100
4	09/05/2021	Examen II	76/100

<

>

Les Redirections. (Routing)

- Le routage est la clé de fonctionnement des SPA.
- À la place d'ouvrir de nouvelles pages, nous avons vu qu' Angular préfère utiliser l'insertion de composant (enfant) dans le parent de sorte que l'on utilise toujours la même page du navigateur.
- Pour insérer un composant, on utilise son sélecteur.
- Problématique: Comment remplacer un composant enfant par un autre dans une barre de navigation (par exemple) sans activer/désactiver plusieurs sélecteurs?
 - On veut que les composants *detail.component* et *edit.component* occupent le même **slot** en remplacement l'un de l'autre.
 - Angular utilise les redirections au sein d'une même page pour gérer ces cas.

Les Redirections. (Routing)

- Lors de la gestion des redirections, un chemin peut être produit dans la barre d'adresse du navigateur.
- Le navigateur est un cas familier d'application de navigation:
 - À la saisie de l'URL dans la barre d'adresse le navigateur ouvre la page correspondante.
 - Le click sur un lien fait naviguer vers une nouvelle page.
 - Le click sur retour/suivant le fait naviguer dans les pages de l'historique de navigation en chargeant/rechargeant les pages.
- La technique de Routing d'Angular évitera de recharger la page principale tout en permettant cette navigation.
- Cette navigation sera possible en réponse à tout autre événement.

Les Redirections. (Étape)

- Définir les Roots dans le *app.module*

```
const routes: Routes = [  
  { path: 'details', component: DetailsComponent },  
  { path: 'edit', component: EditorComponent },  
  { path: '**', component: PageNotFoundComponent },  
];
```

- Chaque « route » indique entre autre:
 - Le chemin du composant qui s'affichera dans la barre de navigation,
 - Le contrôleur du composant vers lequel il mènera.
- Les routes doivent être importées dans un module:
RouterModule.forRoot(routes)
dans les imports de app.module pour les racines

Les Redirections. (Étape)

- Définir le Router Outlet dans le composant de base.
 - Il s'agit d'un composant incarnant le slot (ou placeholder) qui sera remplacé par les composants lors de la navigation.

```
<router-outlet></router-outlet>
```
- Définir le RouterLink qui déclenchera la navigation
 - La demande de navigation peut venir de la barre d'adresse, des boutons de navigation classiques en lien avec l'historique, mais aussi d'événements du UI ou du code d'un contrôleur (programmatique) .

```
<nav>  
  <a [routerLink] ="/details" routerLinkActive="active">Détails</a> |  
  <a [routerLink] ="/edit" routerLinkActive="active">Édit</a> |  
  <a [routerLink] ="/unexisting" routerLinkActive="active">Autre</a>  
</nav>
```
- Lors de la navigation, un certain nombre d'événements et de données transitent dans les routes.

Les Redirections avec Paramètres

- Il est souvent question d'envoyer des paramètres entre les pages de navigation.
 - Il peut s'agir d'id d'objet à éditer,
 - d'autres information de type simples.
- Les routers peuvent accepter des paramètres.
 { **path**: 'edit/:id', **component**: EditorComponent }
- Le *statement expression* associé à la directive *routerlink* aura cette forme
 <a [routerLink] =["'/edit', xxx]">XXX<a>

Exercice d'application

- À partir du lab précédent,
 - Edition d'étudiant
 - Créer un nouveau composant editor permettant d'éditer un étudiant.
 - Avec juste un formulaire.
 - Créer un autre composant blankPage pour afficher les pages non trouvées
 - Navigation
 - Insérer un <outlet-router> dans le template du app.
 - Définissez 3 chemins de navigations menant vers.
 - edit: vers le composant **editor**.
 - details : vers le composant **details** d'étudiants
 - ** : vers un composant page **blankPage**.