

Formation Angular



Angular

- Framework basé sur Typescript, CSS et HTML, développé par Google
- Permet la création d'applications web monopage (SPA)
- Successeur de AngularJS, en 2016

Versions

- Angular 14 (02/06/2022)
 - Composants standalone
 - Liaison sur les attributs protected
 - inject()
- Angular 15 (16/11/2022)
 - host directive
- Angular 16 (03/05/2023)
 - Signals
 - @Input requis
 - takeUntilDestroyed()
- Angular 17 (08/11/2023)
 - Blocs @if @for et @switch directement dans le template

Etude de l'application de démonstration d'Angular

- Installation d'angular

```
npm install -g @angular/cli
```

- Crédit du projet

```
ng new demoApp --skip-tests --style scss --routing false
```

- Pour les premiers exercices, nous allons nous baser sur l'exemple de la version 16 d'Angular

```
npx -p @angular/cli@16 ng new angular-demo --skip-tests --style scss --routing false
```

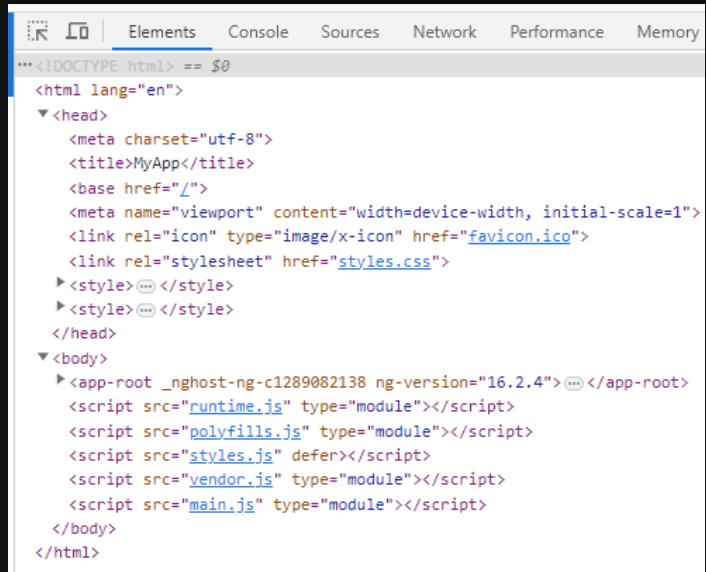
- Lancement

```
ng serve
```

index.html

Comparaison du index.html du projet Angular et du code source du site web généré

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```



The screenshot shows the Chrome DevTools Elements tab with the generated HTML code. The code is identical to the one shown in the code editor, demonstrating that the browser has correctly rendered the Angular application.

```
...<!DOCTYPE html> == $0
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>MyApp</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link rel="stylesheet" href="styles.css">
    <style>@</style>
    <style>@</style>
  </head>
  <body>
    <app-root _ngc="c1289082138" ng-version="16.2.4">@</app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

- vendor.js : bibliothèques externes et dépendances (Angular Material, RxJS, ...)
- polyfill.js : assure la rétrocompatibilité
- styles.js : contient tous les fichiers de style de l'application
- main.js : fichier principal, contient le code source de l'application
- runtime.js : code propre à Angular

```
✓ Browser application bundle generation complete.

Initial Chunk Files | Names      | Raw Size
vendor.js           | vendor     | 2.03 MB
polyfills.js        | polyfills  | 333.17 kB
styles.css, styles.js | styles    | 230.44 kB
main.js             | main       | 47.38 kB
runtime.js          | runtime   | 6.51 kB

| Initial Total | 2.63 MB

Build at: 2023-09-11T09:13:07.875Z - Hash: a37fd99521b672f2 - Time: 4095ms
```

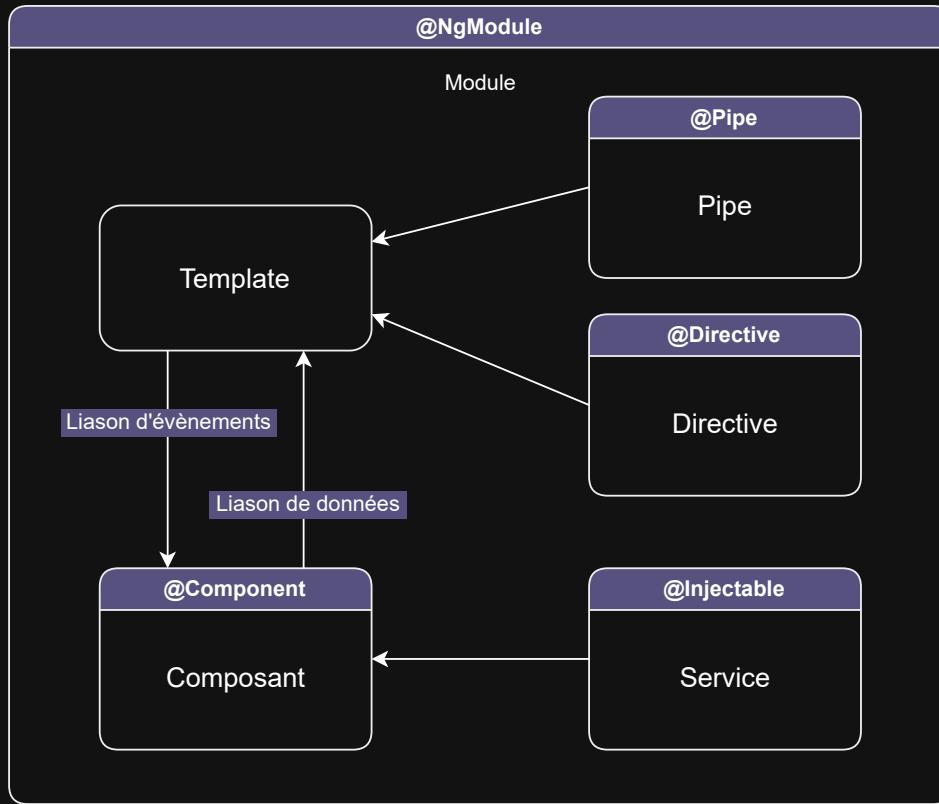
main.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule).catch(err => console.error(err));
```

- Point d'entrée de l'application
- Charge et initialise le module principal de l'application, ici AppModule
- platform-browser-dynamic assure la compatibilité des différents navigateurs

Classes Angular



Modules

- Un module est un regroupement des autres classes Angular
 - Les composants, directives et pipes sont déclarés dans des modules
 - Les modules peuvent exporter une partie de leurs déclarations pour les autres modules
 - Chaque application Angular possède au moins un module, le module racine, utilisé par Angular pour lancer l'application
-
- Depuis Angular 14 et les composants standalones, il est possible de faire une application entièrement sans modules
 - A partir de la version 17, les composants sont standalone par défaut

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Décorateur @NgModule

- Indique à Angular que la classe AppModule est un module
- Utilisation de métadonnées pour paramétriser le module

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Déclarations

- Déclaration de tous les composants, directives et pipes du module
- Les éléments déclarés sont ceux qui appartiennent à ce module, et non ceux exportés

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Modules importés

- Le module BrowserModule contient les services essentiels pour que l'application fonctionne sur navigateur
- Réexporte le module CommonModule, qui contient les directives et pipes basiques de Angular
- Il est également possible d'importer des composants standalone

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Composant racine de l'application

- Au démarrage de l'application, Angular recherche dans le DOM, le premier élément correspondant au sélecteur du composant racine et l'injecte à cet endroit

Composants

- Brique de base d'une application Angular
- A chaque composant est assigné un template

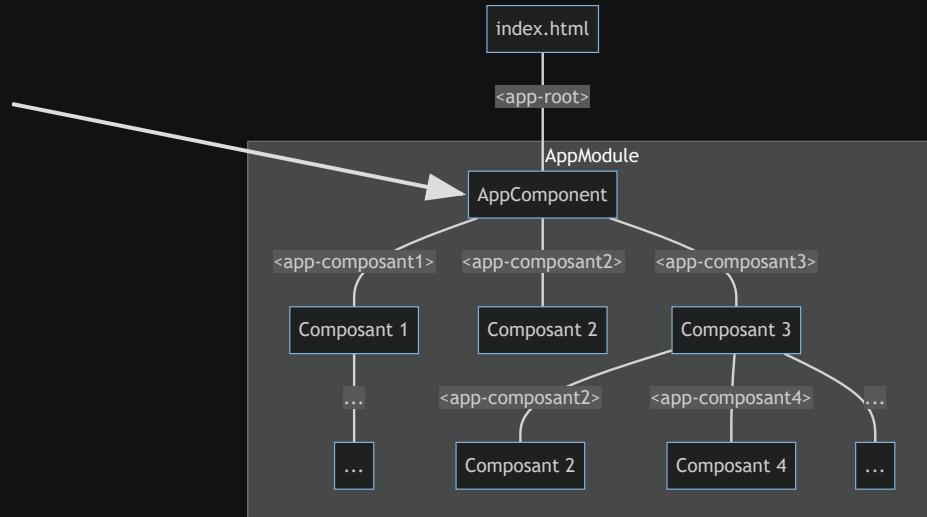
Composant

- Fichier Typescript
- Détermine le comportement du template

Template

- Fichier HTML avec des additions propres à Angular
- Peut avoir son propre fichier de style
- Informe le composant des actions utilisateur

Découpage en composants



- Une application Angular se présente sous la forme d'une arborescence de composants, et contient au moins un composant, le composant racine
- Chaque composant est assigné à un selecteur
- A chaque fois que le selecteur apparaît dans un template, Angular crée une nouvelle instance du composant et l'injecte à cet endroit

Exemple

AppComponent

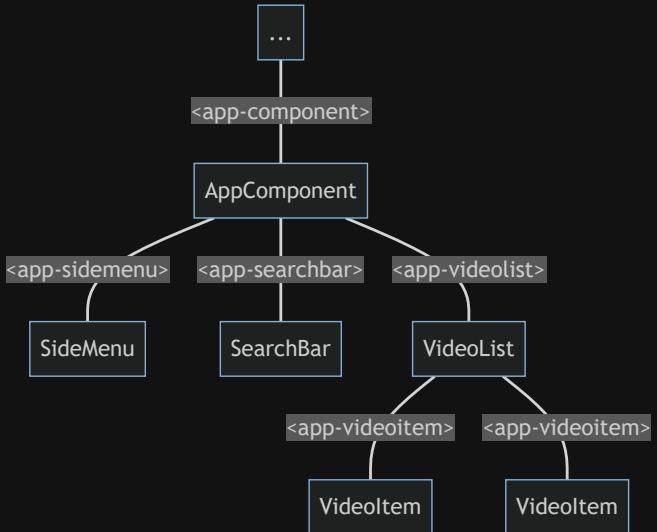
The screenshot shows the YouTube mobile application interface. On the left is a vertical sidebar with a red border containing a navigation menu. At the top right is a search bar with a magnifying glass icon. The main content area displays a grid of video thumbnails. Some thumbnails are highlighted with red boxes, indicating specific components:

- SideMenu**: The sidebar on the left.
- SearchBar**: The search bar at the top right.
- VideoList**: A row of four video thumbnails at the top.
- Videoltem**: Two video thumbnails in the middle row.
- Videoitem**: Two video thumbnails in the middle row.
- VideoList**: A row of four video thumbnails at the bottom.
- Videoltem**: Two video thumbnails in the bottom row.
- Videoltem**: One video thumbnail in the bottom row.

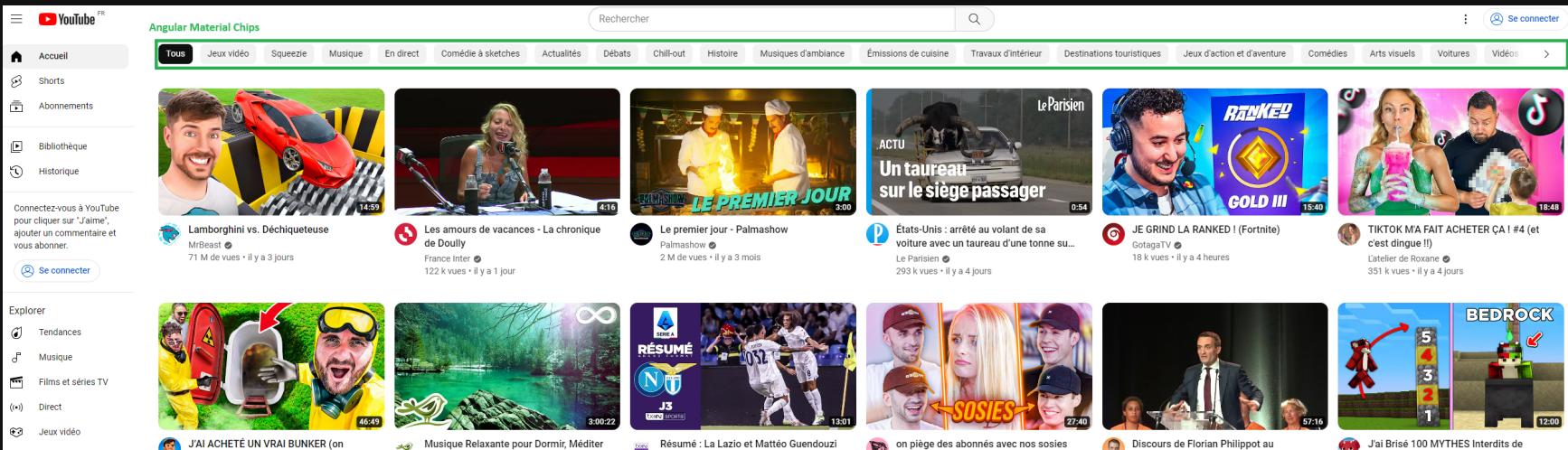
Below the sidebar, there is a message: "Connectez-vous à YouTube pour cliquer sur 'J'aime', ajouter un commentaire et vous abonner." There is also a "Se connecter" button.

On the left sidebar, the following categories are listed:

- Shorts
- Abonnements
- Bibliothèque
- Historique
- Tendances
- Musique
- Films et séries TV
- Direct
- Jeux vidéo
- Actualités
- Sport
- Savoirs & Cultures
- Mode et beauté
- Chaines
- Autres contenus YouTube
- YouTube Premium
- YouTube Music
- YouTube Kids
- Paramètres



Utilisation de bibliothèques



- La bibliothèque Angular Materials (<https://material.angular.io>) fournit des composants réutilisables
- La SNCF possède sa propre bibliothèque de composants

app.component.ts

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

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

Décorateur @Component

- Indique à Angular que la classe AppComponent est un composant
- Utilisation de métadonnées pour paramétriser le composant

app.component.ts

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

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

Sélecteur CSS du composant

Exemple dans index.html

```
...
<body>
  <app-root></app-root>
</body>
...
```

app.component.ts

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

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

Template HTML associé au composant :

- Sous la forme d'un fichier (comme dans cet exemple)
- Sous forme de texte, dans ce cas utiliser l'attribut template à la place

app.component.ts

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

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

Fichier de style utilisé :

- Comme pour le template, soit texte soit fichier
- style et styleUrls sont des tableaux, donc plusieurs entrées possibles
- Par défaut le style est propre au composant
- Un fichier styles.css est présent à la racine pour les styles globaux au projet

app.component.ts

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

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

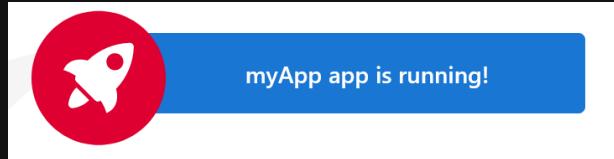
Code du composant

app.component.html

Interpolation

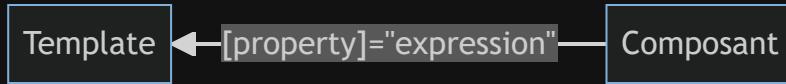


- Dans l'exemple précédent, {{ title }} est remplacé par myApp



- Il y a l'intérieur de l'interpolation une expression de template qui est évaluée et convertie en string
- Il est possible d'accéder aux propriétés et méthodes du composant dans l'expression
- **⚠ Les méthodes dans les expressions doivent être simples, prévisibles et sans effet de bord**
- **⚠ Aux interpolations avec les tableaux et objets**

Property binding



- La syntaxe [] indique à Angular de lier une expression avec une propriété d'un élément du template
- Le lien se fait uniquement du composant vers le template
- Si la propriété du composant change, le template sera modifié en conséquence
- ⚠ Mais pas l'inverse

Template

```
...  
<img [src] = "image" />  
...
```



Composant

```
...  
image = 'https://t.ly/B0f_Z'  
...
```

Attribute binding

- **⚠️** Les attributs HTML sont différents des propriétés du DOM
- Il est possible de lier à un attribut, en préfixant avec "attr"
- Exemple :

```
<input [disabled]="condition">  
<input [attr.disabled]="condition ? 'disabled' : null">
```

- Les deux input ont le même comportement (disabled si condition est vraie)
- **⚠️** La propriété disabled attend un booléen, alors que l'attribut disabled attend un string
- **⚠️** Tous les attributs n'ont pas de propriété correspondante, et inversement
- **⚠️** Il peut y avoir des différences de comportement entre attribut et propriété (value d'un input)
- Privilégiez l'utilisation des propriétés

Class / Style binding

Classe

```
<div [class.class1]="boolean"> ... </div>
```

```
<div [class]="classExpression"> ... </div>
```

```
classExpression = 'class1 class2'
```

```
classExpression = { class1: true, class2: false }  
classExpression = ['class1', 'class2']
```

- Ici le div aura la classe "class1" si l'attribut "boolean" du composant est truthy
- classExpression peut être sous forme de string, de tableau ou d'objet

Style

```
<div [style.fontSize]="fontSize"> ... </div>  
<div [style.fontSize.px]="size"> ... </div>
```

```
<div [style]="styleExpression"> ... </div>
```

```
styleExpression = 'color: blue; font-size: 50px'  
styleExpression = { color: 'blue', 'font-size': '50px' }
```

- Il est possible de lier un style qui a une unité avec un nombre
- styleExpression peut être sous forme de string ou d'objet
- **⚠** Même problème pour les tableaux et objets que l'interpolation

Template expression

- Une expression de template est très similaire à du JavaScript, avec quelques exceptions
 - Il n'est pas possible de faire des assignements (`=, +=, -=, ...`) ni d'incrément/décrément (`++/- -`)
 - Les opérateurs binaires (`|, &`) sont interdits
 - Il n'est pas possible de chainer les expressions avec ; or ,
 - Pas de new, typeof ou instanceof
 - L'opérateur binaire `|` (OR) de JavaScript est remplacé par le pipe
 - La fonction `$any()` pour cast une expression en any
-  Privilégier des expressions simples, avec la logique dans le composant

Exemples

```
 {{ 120 + 10 }}

<pre [style.fontSize.px]="valeurX - 20"> Mon texte </pre>

<pre [style.fontSize.px]="valeurX = 10"> Mon texte </pre>

{{ valeurX += 10 }}

{{ valeurX++ }}

{{ valeurX | date }}

<input [value]="valeurX | pipe" />

{{ objetX }}

{{ getPropertyX() + 10 }}

{{ methodWithSideEffect() }}

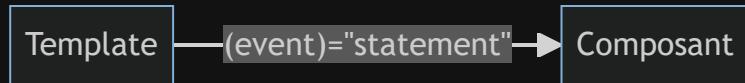
{{ getPropriete() > 12 ? 'Valeur par défaut' : (objetX.propriete | date) + ' est la date' }}

{{ methodeX(valeurX); valeurX }}
```

Exemples

```
 {{ 120 + 10 }} ←— OK →  
 <pre [style.fontSize.px]="valeurX - 20"> Mon texte </pre> ←— OK →  
 <pre [style.fontSize.px]="valeurX = 10"> Mon texte </pre> ←— Opérateur = interdit →  
 {{ valeurX += 10 }} ←— Opérateur += interdit →  
 {{ valeurX++ }} ←— Opérateur ++ interdit →  
 {{ valeurX | date }} ←— OK →  
 <input [value]="valeurX | pipe" /> ←— OK →  
 {{ objetX }} ←— OK →  
 {{ getPropertyX() + 10 }} ←— OK →  
 {{ methodWithSideEffect() }} ←— Attention →  
 {{ getPropriete() > 12 ? 'Valeur par défaut' : (objetX.propriete | date) + ' est la date' }} ←— Attention →  
 {{ methodeX(valeurX); valeurX }} ←— ; interdit →
```

Event binding



- Lie un évènement avec un template statement

```
<button (click)="i=i+1">Incrémenter</button>
```

- Il est possible d'utiliser les méthodes du composant

```
<button (click)="reset()">Réinitialiser</button>
```

- Il est possible de récupérer des informations sur l'évènement grâce à \$event

```
<input (input)="logInput($event)" />
```

```
logInput(event: Event){  
  console.log(event);  
}
```

- Le type de l'objet \$event dépend du type d'évènement, qui sont basés sur l'interface Event
- <https://developer.mozilla.org/fr/docs/Web/API/Event>

Template statement

- Comme les expressions, les templates statements sont proches du JavaScript, avec les différences suivantes :
 - L'assignement basique (=) est permis
 - Le chainage des expressions avec ; ou , est permis
 - La fonction \$any() est autorisée
 - new, typeof et instanceof sont interdits
 - Les opérateurs (+=, -=, ++ et --) sont interdits
 - Les opérateurs binaires sont interdits
 - L'opérateur pipe est interdit

Exemples

```
<button (click)="120 + 10">Test</button>
```

```
<button (click)="valeurX = 10">Test</button>
```

```
<button (click)="valeurX += 10">Test</button>
```

```
<button (click)="valeurX = valeurX | 10">Test</button>
```

```
<button (click)="valeurX | date">Test</button>
```

```
<button (click)="valeurX = objetX?.propriete">Test</button>
```

```
<button (click)="valeurAny = objetAny.propriete">Test</button>
```

```
<button (click)="objetX.propriete = 100">Test</button>
```

```
<button (click)="methodeX(valeurX)">Test</button>
```

```
<button (click)="methodeY($event); methodeX(objetX.propriete)">Test</button>
```

```
<button (click)="methodeX($any($event))">Test</button>
```

```
<button (click)="console.log(valeurX)">Test</button>
```

Exemples

```
<button (click)="120 + 10">Test</button> ←!— OK, mais ne fait rien →
```

```
<button (click)="valeurX = 10">Test</button> ←!— OK →
```

```
<button (click)="valeurX += 10">Test</button> ←!— += interdit →
```

```
<button (click)="valeurX = valeurX | 10">Test</button> ←!— | interdit →
```

```
<button (click)="valeurX | date">Test</button> ←!— | interdit →
```

```
<button (click)="valeurX = objetX?.propriete">Test</button> ←!— Erreur →
```

```
<button (click)="valeurAny = objetAny.propriete">Test</button> ←!— Erreur à l'execution →
```

```
<button (click)="objetX.propriete = 100">Test</button> ←!— OK →
```

```
<button (click)="methodeX(valeurX)">Test</button> ←!— OK, mais ne fait rien →
```

```
<button (click)="methodeY($event); methodeX(objetX.propriete)">Test</button> ←!— OK →
```

```
<button (click)="methodeX($any($event))">Test</button> ←!— OK →
```

```
<button (click)="console.log(valeurX)">Test</button> ←!— console n'est pas accessible →
```