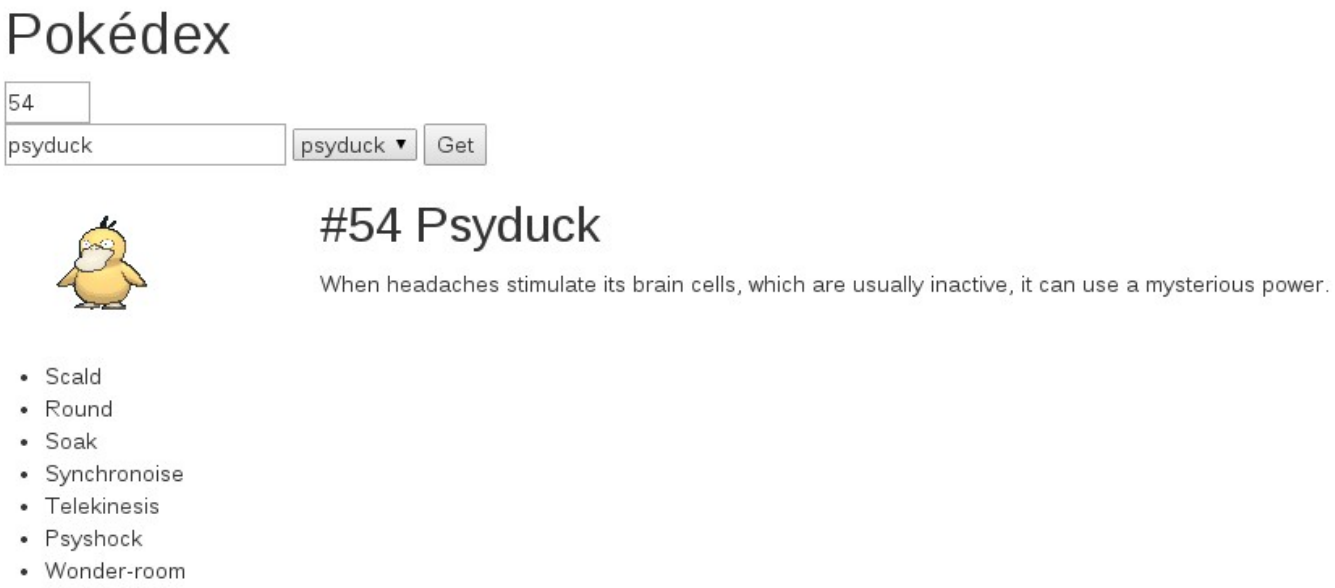
Angular 2

Un jour je serai le meilleur codeur
Je développerai sans répit
Je ferai tout pour être vainqueur
Et gagner les défis
Je parcourrai le web entier
Traquant avec espoir
Les frameworks et leurs mystères
Le secret de leurs pouvoirs

[Refrain]
Angular !
Développez les tous
C'est notre TP
Ensemble pour la victoire
Angular !
Rien ne nous arrêtera
Notre binôme triomphera

Votre aventure dans le monde merveilleux des frameworks JavaScript commence avec Angular. Angular est un framework développé par Google. Ce framework se base principalement sur la définition de composants et de services. L'objectif est d'avoir un code modulaire et réutilisable. Le langage recommandé pour programmer avec Angular et celui que nous allons utiliser est Typescript. On peut aussi utiliser Javascript 5/6 ou bien Dart.

Pour vous aider dans votre quête, le professeur Chen vous a laissé des instructions ainsi que les éléments de bases pour créer un Pokédex à cette adresse : https://github.com/gbecan/teaching-jxs-angular2 (l'utilisation d'une bicyclette est conseillée pour s'y rendre plus vite).



Recherche d'un pokémon via son numéro

La première étape pour développer notre pokédex consiste à proposer au dresseur de rechercher un pokémon via son numéro. Nous allons donc créer un composant Angular qui va contenir et gérer cette interface. Pour créer un composant, il suffit de créer une classe et de lui ajouter l'annotation @Component. Cette annotation peut être complétée par plusieurs paramètres. Pour le moment nous allons en utiliser trois :

- **selector** : permet de définir le nom de l'élément html qui sera remplacé par notre composant
- **templateUrl** :permet de spécifier le fichier html qui servira de vue au composant
- **directives** : permet d'utiliser d'autres composants ou directives dans le template du composant courant

```
@Component({
     selector: "my-element",
     templateUrl: "app/my-component.html"
})
export class MyComponent {

}
```

Q1 : Créer un composant avec un élément <*input* > afin de récupérer l'id recherché. Ajouter votre composant à la liste des directives et au template du composant PokedexComponent.

Nous allons maintenant utiliser le data-binding d'Angular pour lier l'élément <*input*>à un attribut de notre composant. En quelque sorte, nous allons lier la vue à notre modèle. Pour cela, on utilise ngModel qui s'utilise comme ceci :

```
<input [(ngModel)]="id">
```

Ce code lie la valeur de l'élément input à l'attribut *id* de notre composant.

Q2 : Créer un attribut *id* et lier le à l'élément <input> précédemment créé.

Pour tester le lien entre l'attribut et l'élément HTML, nous allons utiliser une autre syntaxe utilisant aussi le méchanisme de data-binding : {{myAttribute}}. Cette syntaxe permet d'afficher la valeur d'un attribut du composant sur la page.

Q3 : Afficher la valeur de l'id renseigné dans la balise <input> précédente.

Recherche dans une liste

Malheureusement, seul le professeur Chen connaît précisément le numéro de tous les pokémons. Pour aider les jeunes dresseurs à utiliser le pokédex, nous allons offrir la liste des pokémons ainsi qu'un champ de recherche pour filtrer cette liste.

Q4 : Créer une classe Pokemon qui comporte un id et un nom. Nous compléterons la classe au fur et à mesure du TP.

Q5 : Créer une liste fictive (4-5 éléments suffiront) de pokémons dans le composant précédemment créé.

Q6 : Afficher la liste des pokémons dans une balise <select> en utilisant ngFor (https://angular.io/docs/ts/latest/guide/template-syntax.html#!#ngFor)

Q7 : Récupérer le choix du dresseur en liant la balise <select> au modèle avec ngModel.

Comme la liste des pokémons peut être très longue, nous allons proposer au dresseur de filtrer la liste.

Q8 : Ajouter un champ de texte et récupérer sa valeur dans un attribut. Utiliser le filtre PokemonNamePipe fournit pour filter la liste des pokémons suivant la valeur du champ texte. Pour pouvoir utiliser ce filtre, il faut ajouter un paramètre *pipes* au composant : `pipes: [PokemonNamePipe]`

Pour valider le choix du dresseur, nous allons ajouter un bouton « Go ! » dont le comportement sera défini dans notre composant. Pour lier un évènement à notre composant, on utilise la syntaxe (eventName).

Q9 : Ajouter un <button> à la page et lier l'évènement *click* à une méthode du contrôleur. Pour le moment la méthode se contentra d'afficher l'id ou le nom du pokémon recherché dans la console.

Accès à une API

Le site http://pokeapi.co/ propose une API contenant de nombreuses informations sur les pokémons. En particulier, l'API offre la liste des pokémons (api/v2/pokedex/1) ainsi que des informations détaillées pour chacun d'entre eux (api/v2/pokemon/54 ou api/v2/pokemon/psyduck). Nous allons utiliser cette API comme source d'information pour notre pokédex.

Angular fournit un service HTTP qui va nous permettre de communiquer avec PokéAPI. Angular utilise l'injection de dépendances pour fournir les services. Cela permet en particulier d'instancier un service qu'une seule fois pour toute l'application ou bien une partie de celle-ci. Pour encapsuler l'accès à l'API, nous allons nous même créer un service. Un service Angular est en fait une classe classe.

Q10 : Créer un service pour gérer l'accès à PokéAPI. Ajouter un paramètre à son constructeur afin de récupérer le service http. Ajouter aussi l'annotation @Injectable() afin de spécifier à Angular que votre service comporte une dépendance vers un autre service.

Q11 : Créer une méthode pour récupérer la liste des pokémons en utilisant le service http.

Q12 : Utiliser ce service dans le composant de recherche de pokémons pour remplacer la liste fictive de pokémons. Pour cela, ajouter le paramètre *providers* au composant PokedexComponent comme ceci : `providers: [PokeApiService]`. Ainsi Angular saura quelle classe injecter dans vos composants.

Q13 : Créer une autre méthode pour récupérer les informations sur un pokémon. Compléter la classe Pokémon afin de recueillir ces informations.

Q14 : Créer un nouveau composant dédié à l'affichage des informations d'un pokémon. Utiliser le service précédemment créé pour récupérer les informations d'un pokémon. Combiner les différents méchanismes de data-binding vus jusqu'ici pour afficher l'id, le nom et les statistiques d'un pokémon.

Communication entre contrôleurs

À présent, nous avons deux parties à notre application. La première permet de rechercher un pokémon grâce à son numéro ou son nom. La deuxième récupère et affiche les informations d'un pokémon. Il ne reste plus qu'à relier ces deux parties pour finaliser notre pokédex. Pour faire communiquer nos deux composants, nous allons créer un nouveau service qui va contenir les informations à partager. Comme pour le service dédié à l'API, ce service ne sera instancié qu'une seule fois et permettra donc l'échange d'informations.

Q15 : Créer un service contenant l'id du pokémon recherché. Injecter ce service dans le composant de recherche et le composant d'affichage des informations. On n'oublira pas de l'ajouter dans les providers du composant PokedexComponent.

Les deux composants ont maintenant un service pour partager des informations liés et utilisent les mêmes informations. Cependant, les informations du pokémon ne sont pas mises à jour si le dresseur change le numéro ou le nom du pokémon recherché. Pour détecter les changements de ces deux attributs, nous allons utiliser la notion d'observable. Un exemple est disponible à cet adresse : https://coryrylan.com/blog/angular-2-observable-data-services

Q16 : Créer un observable dans le service précédemment créé. Le composant d'affichage d'informations d'un pokémon peut maintenant souscrire à cet observable pour détecter le changement de pokémon.