OEFENINGEN ANGULAR FUNDAMENTALS

01 – EENVOUDIG BEGINNEN

a) Het algemene Github-adres met voorbeeldcode is github.com/PeterKassenaar/voorbeeldenAngular2

b) Maak/start een eigen Hello World-app. Dit kan op verschillende manieren:

- 1. Gebruik als basis `100-helloworld`.
  - Voer hiervoor een `npm install` en `npm start` uit.
  - Open het project in je browser op http://localhost:4200. Bekijk de code.
- 2. OF: ga zelf naar `cli.angular.io`, installeer deze tool en maak een nieuw project.
  - Neem de standaardantwoorden van de Angular CLI-prompt over door steeds op `Enter` te drukken.
  - Lees hiervoor zelf de online instructies (zie ook 1d.). Je vindt deze ook in de Angular QuickStart, op https://angular.io/guide/quickstart.

c) Optioneel: (dit doe je *niet* in echte applicaties): Probeer eens of je een nieuwe component kunt maken en deze bootstrappen. De nieuwe component komt dan *in plaats van* `<hello-world>`, nog niet erbij. Volg deze stappen:

- *Handmatig?* - maak een nieuw bestand, bijvoorbeeld `\app\nieuwe.component.ts`.
  - Importeer de juiste afhankelijkheden.
  - Schrijf een selector en een HTML-template.
- *CLI?* - gebruik de opdracht `ng generate component <component-naam>`.
- Pas `app.module.ts` aan, zodat daarin nieuwe component wordt geïmporteerd en gestart.
- Pas `index.html aan`, zodat de juiste selector wordt gebruikt.
- Draai `npm start` om je nieuwe component te testen.

d) Installeer Angular-CLI en maak hiermee een basisproject.

- De aanwijzingen hiervoor vind je op https://cli.angular.io/ .
- Maak een nieuw project met `ng new <projectnaam>`.
- Gebruik de opdrachten `ng new`, `ng serve` en `ng generate`. Zoek zelf op hoe dit werkt.

e) Optioneel, om kennis te maken TypeScript : ga naar `typescriptlang.org` en oefen met het maken van een Class. Maak bijvoorbeeld een klasse `Person`, met properties als `firstName`, `lastName` en `email`. Test hoe deze properties worden bereikt door ze in de console te loggen.

- Bekijk ook andere voorbeelden uit het dropdown-menu (zoals `Types`, `Generics`, `Modules`) om breder kennis te maken met TypeScript.

02 – DATABINDING

a) Breid je app uit oefening 1 (`Hello World`) uit met een field/property. Bindt deze property in het template dat gebruikt wordt.

- Doe dit zowel impliciet (met rechtstreekse initialisatie van variabelen) als expliciet (met aparte declaratie en initialisatie in de `constructor`).
- Gebruik daarna `ngOnInit()` voor initialisatie van je variabele.
- Lees de werking en de volgorde van Angular lifecycle hooks op https://angular.io/guide/lifecycle-hooks.
- Codevoorbeeld: `101-databinding`.

b) Maak een array van objecten. Bind ze in het template met de directive `*ngFor`.
- Geef via TypeScript expliciet op dat de property uit een array van strings of een array van objecten moet bestaan (maak zelf deze keuze).
- Codevoorbeeld: `101-databinding`.

c) Maak expliciet een `Model` voor de inhoud van je array. Dat mag een object met één of meerdere properties zijn. Zorg er voor dat de inhoud van de array bestaat uit objecten van het type `<Model>`.
- Codevoorbeeld: `101-databinding` - `citiesVolgensModel`
- Optioneel: in plaats van een class te gebruiken voor het `Model` mag je ook het TypeScript-construct `Interface` gebruiken. Zoek eventueel zelf op hoe dit werkt.

d) Bedenk zelf een toepassing voor de directive `*ngIf` en gebruik deze in combinatie met je class en properties.

e) Breng de template naar een externe .html-file. Gebruik vervolgens in `@Component` de property `templateUrl` om hier naar te verwijzen. Test of de databinding blijft werken.

03 – EVENT BINDING

a) Voeg een element met event-binding toe aan je applicatie. Bijvoorbeeld een `(click)` op een knop afvangen. Roep een event handler in de component aan als de event optreedt. Laat deze bijvoorbeeld een `alert()` tonen of een `console.log()`-melding.
- Codevoorbeeld: `102-eventbinding`.

b) Test meerdere typen DOM-events, bijvoorbeeld `blur`, `focus`, `keypress`, `mousemove`, enzovoort. Zie eventueel Mozilla.org (https://developer.mozilla.org/en-US/docs/Web/Events) voor een overzicht.

c) Optioneel: Test een van de niet-DOM events, bijvoorbeeld de networking events `online` en `offline`.
- Schrijf hiervoor de decorator `@HostListener('window:offline')` en een eventHandler-function.
- Test of de event goed werkt door het `offline`-event in de browser te simuleren.
- Optioneel : gebruik een van de andere events die je in de lijst bij MDN hebt gezien (denk bijvoorbeeld aan het `scroll` event, `resize` event of iets met de batterij. Let op: niet alle events werken op dezelfde manier, of zijn zo eenvoudig!)

d) Maak een formulierveld (bijvoorbeeld een tekstveld) met een local template variable. De notatie hiervoor is `#myFieldName`. Geef de variabele door aan de event handler en verwerk hem in een `alert()`, `console.log()`-melding, om een waarde in het DOM te updaten, enzovoort.

e) Maak een eenvoudige client-sided CRUD-applicatie: users kunnen elementen toevoegen aan een array (namen, producten, enzovoort) en verwijderen uit de array.
   - Gebruik voor verwijderen de JavaScript-functie `ArrayName.splice(…)` of `ArrayName = ArrayName.filter(item => …)`.
   - Codevoorbeeld: `102-eventbinding\...\app-02-complete.component.ts`.

04 - ATTRIBUTE BINDING EN TWO-WAY BINDING

a) Maak een component met een tekstvak. Als de user een (Engelstalige) kleur in het tekstvak invult en op een knop klikt, krijgt een bijbehorende `<div>` deze achtergrondkleur.
   - Tip: gebruik `<div [style.background-color]="someVariable">` als attribuut om de achtergrondkleur in te stellen.
   - Breid dit eventueel uit met een tweede tekstvak voor het instellen van de tekst/voorgrondkleur.
   - Optioneel: onderzoek hoe dit werkt als de kleuren uit een serie radio buttons gekozen kunnen worden, of uit een selectielijst/dropdown.
   - Algemeen codevoorbeeld attribute binding: `103-attributebinding`.

b) Optioneel: onderzoek zelf hoe de Angular-concepten *class binding* en *style binding* worden gebruikt. Aanwijzingen hiervoor staan op https://angular.io/guide/template-syntax#ngclass.

c) Maak een tekstveld in je component met two-way binding. De syntaxis hiervoor is `[(ngModel)]`. Toon de waarde van de ingevoerde tekst in de pagina.
   - Denk aan het toevoegen van `FormsModule` aan `app.module.ts`.
   - Maak een kopieerfunctie: Maak twee tekstvakken op de pagina. Tekst die in het ene tekstvak wordt ingevuld, verschijnt ook in het andere tekstvak.
   - Codevoorbeeld: `104-twowaybinding`.

05 - SERVICES

a) Als het goed is heb je inmiddels enkele componenten met hierin diverse data. Verplaats deze data vanuit de component naar een service.
   - Maak eerst de service via `ng generate service [naam]`.
   - Injecteer de service in de constructor van de component waarin je de data wilt gebruiken.
   - Denk aan de property `[providers]` in de `@ngModule`-annotatie.
   - OF: gebruik de notatie `@Injectable({ providedIn: 'root' })`. Dit is de standaardwijze die Angular CLI gebruikt sinds Angular 6.
   - Codevoorbeeld : `200-services-static`.

b) Optioneel: probeer eens (in echte applicaties: NIET aanbevolen) om de service alleen in de *component* te injecteren via `providers:[]` in `app.component.ts`. Test of alles nog net zo werkt als voorheen. In echte applicaties is dit niet aanbevolen, omdat op deze wijze iedere component *zijn eigen* instantie van de service krijgt. De service is dan geen *singleton* meer.

c) Werken met async-services: maak zelf een `.json`-bestand met data en laadt dit in je applicatie. Denk onder meer aan:
   - het injecteren van `HttpClientModule` in de module;
   - het aanpassen van de component waarin de service wordt geconsumeerd: `private http:HttpClient` injecteren in de constructor en de observable-notatie met `.subscribe()` gebruiken.
   - Codevoorbeelden: `0201-services-http` en `202-services-rxjs`.

d) Meer over RxJs: gebruik ook andere operators binnen `.pipe()`. Test bijvoorbeeld:
   - `map()`, `delay()`, eventueel `filter()` en andere.
   - Codevoorbeeld : `202-services-rxjs`.

e) Maak je toepassing geschikt voor het werken met de `async pipe`.
   - Codevoorbeeld: `206-services-async-pipe`

f) Werken met Live API's: gebruik een API naar keuze om live data op te halen. Maak een nieuwe applicatie en gebruik hierin deze data.
   - API's zijn bijvoorbeeld beschikbaar op `pokeapi.co`, `openweathermap.org/API` of `www.omdbapi.com`.
   - Zie ook de site https://github.com/public-apis/public-apis en het bestand `JavaScript APIs.txt` voor meer API-endpoints.
   - Codevoorbeeld: `210-services-live`.

06 - APPLICATIES MET MEERDERE COMPONENTEN

a) Maak een nieuwe applicatie, of ga uit van je applicatie uit de eerdere oefeningen.

b) Voeg een detail-component toe. De details worden getoond na een muisklik op de hoofdcomponent.
   - Volg het stappenplan (4 stappen) uit de presentatie, waarin de component wordt gemaakt, ingesloten in de hoofdcomponent en op de juiste wijze in HTML wordt aangeroepen.
   - OF: gebruik de CLI om nieuwe component te genereren. Gebruik de selector om hem in te voegen in de HTML van AppComponent (bijvoorbeeld `<mijn-nieuwe-component></mijn-nieuwe-component>`).
   - Codevoorbeeld: `300-components`.

c) Gebruik de annotatie `@Input()` om data vanuit de hoofdcomponent door te geven aan de detailcomponent.
   - Zorg er voor dat een object wordt doorgegeven aan de detailcomponent. Gebruik hiervoor de notatie `[objName]="objData"` in de template van de hoofdcomponent.
   - Toon de inhoud van het object in de detailcomponent.
   - Codevoorbeeld: `301-components-inputs`.

d) Gebruik de annotatie `@Output()` om events vanuit de detailcomponent door te geven aan de hoofdcomponent. Denk bijvoorbeeld aan het doorgeven van een rating (zoals in het voorbeeld), of iets anders.

- Codevoorbeeld: `302-components-outputs`.

e) Complete applicatie: maak een eenvoudige eCommerce-applicatie met de volgende onderdelen:
   - Een Store. Schrijf een service voor de store die een 4 of 5 producten levert (dit mag statische data zijn).
   - Een Detailcomponent. Bij klikken op een product worden details over het product getoond.
   - Een Shopping Cart. De user kan het product in een winkelmandje plaatsen. De inhoud van het winkelmandje is steeds zichtbaar in de hoofdcomponent.
   - Een "Order"-knop. Als hierop wordt geklikt, wordt de inhoud van het winkelmandje getoond en het totaalbedrag berekend.
   - Codevoorbeeld: `303-pubsub-ordercomponent`.

07 - ROUTING

a) Maak je applicatie uit de vorige oefeningen geschikt voor routing, of begin een nieuwe applicatie. De eisen zijn de volgende:
   - Zorg er voor dat de applicatie een rootComponent heeft, waarin het "hoofdmenu" met opties wordt getoond.
   - Zorg er voor dat verschillende componenten worden geladen als een keuze wordt gemaakt in het hoofdmenu.
   - Codevoorbeeld: `400-router`.

b) Optioneel: maak een geheel nieuwe applicatie via de CLI en stel direct in dat deze applicatie routing gebruikt. De syntaxis hiervoor is `ng new <applicatie> --routing`.
   - Als de flag `--routing` niet is gebruikt, beantwoord dan de vragen over routing in de CLI-prompt (kies `(Y)es` bij de vraag of je routing wilt gebruiken).
   - Bekijk hoe de CLI een `app-routing.module.ts` heeft gegenereerd en wat de syntaxis hierbij is. Bekijk ook hoe deze wordt gebruikt in de hoofmodule `app.module.ts`.
   - Lees eventueel https://github.com/angular/angular-cli/wiki voor meer informatie.

c) Maak je applicatie geschikt voor het werken met route-parameters. Denk bijvoorbeeld aan het maken van een detailroute voor een bepaald product in je applicatie. Denk aan:
   - het uitbreiden van `app.routes.ts` met parameters;
   - het insluiten van `ActivatedRoute` om de parameters te kunnen oppikken in de detailcomponent;
   - Het aanpassen van de HTML om parameters te kunnen meegeven (`[routerLink]=…`)
   - In de detailcomponent gegevens opvragen op basis van de meegegeven parameter.
   - Codevoorbeeld: `401-router-parameter`.

08 - FORMULIEREN (- TODO: VERTALEN NAAR NL)

a) Goal: build a template driven form

- Create a simple HTML5 form in a component, or use a form from an earlier exercise (for example the login form from `11-post-demo`).
- Add the local template variable `#myForm="ngForm"` to the `<form>` tag.
- Add the directives `ngModel` to the separate form fields. You don't need two-way databinding with `[()]`.
- Write for example `myForm.value` to the user interface, or show the contents of the form in an alert (or in the console) when a button is clicked.
- Demo code available at `/500-forms-template-driven`, `Component 1`.

b) Goal: address individual controls inside the form and add HTML5 validators.

- Assign a local template variable to the form fields.
- Bind `ngModel` to the local template variable. The code can look like
  `#email="ngModel"`
- Retrieve the values from the local template variable and show them in the user interface, for example its value and its validity.
- Add the HTML5 attribute `required` to the form fields and see how this affects the state of the form field. Write its validity to the user interface.
- Demo code available at `/500-forms-template-driven`, `Component 2`.

c) Goal: combining individual form fields to an ngModelGroup

- Add some field to the form (for example some extra text fields or checkboxes).
- Groep them inside a `<div>`, assign the `<div>` the directive `ngModelGroup`. The code can look like
  `<div ngModelGroup="customer" #customer="ngModelGroup">`
- Run the code and identify the model group in the returned form value object.
- Optional: set the value of a form field from inside your class, by using the local template variable and bind to `[ngModel]`.
- Demo code available at `/500-forms-template-driven`, `Component 3`.

d) Goal: submitting template driven forms

- Add a submit button to the form.
- Make sure the submit button is only active when the form as a whole is valid. Your code can look like
  `<button type="submit" (click)="onSubmit(myForm)"`
  `[disabled]="!myForm.valid"> … </button>`
- Demo code available at `/500-forms-template-driven`, `Component 4`.

e) Optional goal: working with model driven forms

- Start with a simple form, for example build a form on your own, or use the dummy login form from `/212-post-demo`.
- Import `ReactiveFormsModule` into your `app.module.ts`.

- Add the `[formGroup]="…"` directive to the `<form>` tag, add `formControlName="…"` to the individual controls.
- Import `FormGroup` and `FormBuilder` into your class and build the form, based on the layout of your HTML.
- Submit the form and write the value to an alert box or to the console.
- Demo code available at `/501-forms-model-driven`, `Component 1` and `Component 2`.