Question 1:

Which of the following statements about components in Angular are correct?

(multiple correct answers possible)

- [ ] The properties of a component's children are available in the component's constructor.
- [ ] When a component depends on a service, the injector can be used to configure dependency injection.
- [ ] A component defines its input parameters with the @Input decorator.
- [ ] A component is a type of directive and as such, should be defined by the Directive class decorator.
- [ ] A component selector must reference the class name in lowercase hyphenated format.
- [ ] A components ngOnDestroy method is called just before Angular destroys the component.

Question 2:

Consider the following component, which can be used to model an animal and the noise it makes.

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

@Component({
  selector: 'animal-noise',
  template: `
    <span>{{animal}}</span>
    <button (click)="makeNoise()">Make noise</button>
  `
})
export class AnimalNoise {
  @Input('animal') animal: string;
  @Input('noise') noise: string;

  makeNoise() {
    alert(`${this.noise}`);
  }
}
```

Select the statements about the *AnimalNoise* component that are correct.

(multiple correct answers possible)

- [ ] Component, Input and Output are all required imports for this component.
- [ ] Both animal and noise inputs must be provided when including the AnimalNoise component in a template.
- [ ] The 'animal' parameter of the @Input('animal') declaration does not alter the interface of the component.
- [ ] When included in a components template, the AnimalNoise component creates a span containing the interpolated animals name and a button bound to *makeNoise()*.
- [ ] The AnimalNoise component can be included in another template using the <AnimalNoise> tag.

Question 3:

The following code initializes three types representing the colors: red, green and blue.

```
type ColorType = [string, number, number, number];
let red: ColorType = ['Red', 1, 0, 0];
let green: [string, number, number, number] = ['Green', 0, 1, 0];
let blue = ['Blue', 0, 0, 1];
```

Select the statements that are correct.

(multiple correct answers possible)

- [ ] All three colors can have new elements of any type appended to them.
- [ ] ColorType is an alias to a tuple.
- [ ] All three colors would compile to the same JavaScript type signature.
- [ ] ColorType is an array where the type of a fixed number of elements is known.
- [ ] All three variables are immutable.

Question 4:

The following code initializes strings as three different types and attempts to make them uppercase. Which statements about the behavior of the typescript compiler are correct?

```
let stringType: string = "string type";
stringType.toUpperCase();

let anyType: any = "any type";
anyType.toUpper();
anyType.toUpperCase();

let objectType: Object = "object type";
objectType.toUpperCase();
```

(multiple correct answers possible)

- [ ] The typescript compiler confirms that toUpperCase exists on the stringType instance.
- [ ] The typescript compiler states that toUpper does not exist on the anyType instance;
- [ ] The typescript compiler confirms that toUpperCase exists on the anyType instance.
- [ ] The typescript compiler states that the toUpperCase function does not exist on the objectType instance.
- [ ] The typescript compiler confirms that the toUpperCase function exists on the objectType instance.

5. Start building:
Create a basic calculator Angular app. The app needs to have a header for a logo and a menu. On the first screen of this app, the app selects a random calculation from a database of 5 different. For example a calculation could be: 3 * 6 . You can make the 5 calculations as complicated as you want. You can use a database of choice. Also store a hash of each calculation in the database.

In the interface, list the 5 calculations and highlight the selected calculation per session. Besides the calculation there should be an input field in the database, which the user can use to give the answer to the calculation. In the example above, the answer should be 18. Next to the input field there should be a submit button to send in an answer.

If the user submits the submit button and the answer is correct, send the user to a thank you page in the app (use a funny GIF on the thank you page). From the thank you page he can go to the default page again to get a new random calculation and go again.

If the answer is wrong, the user should get a nice notification that his answer is not correct and that he should try again. The maximum amount of tries is 3. If the user tries more than 3 times, the user failed and is not allowed to try again.

Make sure you store in the database how many tries a user does before he gets the answer correct. Also make sure to store, which questions the user got right or wrong. The webpage can be used by multiple people.

6. Implement the Klippa API
At Klippa we offer APIs and SDKs to work with smart document processing. Companies use our API to process receipts and invoices more efficiently.

Your last task is to build a very basic webpage that allows to upload a picture of a receipt or invoice, submit that to our API, cover the loading time and then show the reponse of our API in a nicely structured table view that contains the invoice data. Make sure it has a 'try again' button to upload multiple times. You can build this webpage in a programming language of choice.

You can find our API documentation on the page below. You will receive an API key with 100 testing credits to use via email.

https://custom-ocr.klippa.com/docs