**JavaScript Developer Answers Time 1 Hour**

1. How to call base class constructor from child class in TypeScript?
2. **Super()**
3. State true/false – below question

<!—Code Start -->

*interface NotEmpty<T> {*

*data: T;*

*}*

*let x: NotEmpty<number>;*

*let y: NotEmpty<string>;*

*x = y;*

<!—code end -->

1. No
2. Choose the right answer :-

*function Foo() {*

*return this;*

*}*

*Foo() === window;*

1. True
2. State true/false – below question

<!—code end -->

*class Animal {*

*feet: number;*

*constructor(name: string, numFeet: number) { }*

*}*

*class Size {*

*feet: number;*

*constructor(numFeet: number) { }*

*}*

*let a: Animal;*

*let s: Size;*

*a = s;*

*s = a;*

<!—code end -->

What is output of the above **a =s, s=a?**

1. True, True
2. Choose the right answer

*var age = {*

*prop: 25,*

*f: function() {*

*return prop;*

*}*

*};*

b)25

1. What is the result of the following statement: typeof “x”;

d) “string”

7. Primitive types are passed by :

1. Value
2. What is Angular hook life cycle? explain it.   
    Angular 2 Lifecycle Events Log:-

1.onChanges

2.onInit

3.doCheck

4.afterContentInit

5.afterContentChecked

6.afterViewInit

7.afterViewChecked

8.doCheck

9.afterContentChecked

10. afterViewChecked

11. onChanges

12. doCheck

13. afterContentChecked

14. afterViewChecked

15. onDestroy

With Component (interface wise) -

1.ngOnChanges - called when an input binding value changes.

2.ngOnInit - after the first ngOnChanges.

3.ngDoCheck - after every run of change detection.

4.ngAfterContentInit - after component content initialized.

5.ngAfterContentChecked - after every check of component content.

6.ngAfterViewInit - after component's view(s) are initialized.

7.ngAfterViewChecked - after every check of a component's view(s).

8.ngOnDestroy - just before the component is destroyed.

1. Does TypeScript supports function overloading? If yes, Explain with a example

class Person {

name: string;

Id: number;

add(Id: number);

add(name:string);

add(value: any) {

if (value && typeof value == "number") {

//write here logic

}

if (value && typeof value == "string") {

//write here logic

}

}

}

(or)

add(a:string|number) {

//write here logic

}

1. What is generics in Angular/ Type Script ? Explain with a example .

-we need a way of capturing the type of the argument in such a way that we can also use it to denote what is being returned. Here, we will use a type variable, a special kind of variable that works on types rather than values.

function identity<T>(arg: T): T {

return arg;

}

Uses -

1. let output = identity<string>("myString"); // type of output will be 'string'

2. let output = identity<number>(10); // type of output will be 'Number'

1. Explain **for..of** vs **for..in** with a help of example?

-for..of and for..in statements iterate over lists; the values iterated on are different though, for..in returns a list of keys on the object being iterated, whereas for..of returns a list of values of the numeric properties of the object being iterated.

let list = [1, 2, 3];

for (let i in list) {

console.log(i); // "0", "1", "2",

}

for (let i of list) {

console.log(i); // "1", "2", "3"

}

1. What is event propagation? Give a example

Event bubbling and capturing are two ways of event propagation in the HTML DOM API, when an event occurs in an element inside another element, and both elements have registered a handle for that event. The event propagation mode determines in which order the elements receive the event.

With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

With capturing, the event is first captured by the outermost element and propagated to the inner elements.

Capturing is also called "trickling", which helps remember the propagation order

1. What is event delegation, DOM life cycle? Give example for each.

Capturing and bubbling allow to implement one of most powerful event handling patterns called event delegation.

The idea is that if we have a lot of elements handled in a similar way, then instead of assigning a handler to each of them – we put a single handler on their common ancestor.

In the handler we get event.target, see where the event actually happened and handle it.

onClick(event) {

let action = event.target.dataset.action;

if (action) {

this[action]();

}

1. Write a Angular “Service” with GET, POST,PUT,DELETE methods? and mention all require libraries.

**import { Injectable } from '@angular/core';**

**import { HttpClient } from '@angular/common/http';**

**import { Observable } from 'rxjs/Observable';**

**import { User } from '../shared/models/user.model';**

**@Injectable()**

**export class UserService {**

**constructor(private http: HttpClient) { }**

**register(user: User): Observable<User> {**

**return this.http.post<User>('/api/user', user);**

**}**

**login(credentials): Observable<any> {**

**return this.http.post<any>('/api/login', credentials);**

**}**

**getUsers(): Observable<User[]> {**

**return this.http.get<User[]>('/api/users');**

**}**

**countUsers(): Observable<number> {**

**return this.http.get<number>('/api/users/count');**

**}**

**addUser(user: User): Observable<User> {**

**return this.http.post<User>('/api/user', user);**

**}**

**getUser(user: User): Observable<User> {**

**return this.http.get<User>(`/api/user/${user.\_id}`);**

**}**

**editUser(user: User): Observable<string> {**

**return this.http.put(`/api/user/${user.\_id}`, user, { responseType: 'text' });**

**}**

**deleteUser(user: User): Observable<string> {**

**return this.http.delete(`/api/user/${user.\_id}`, { responseType: 'text' });**

**}**

**}**

1. What is @Inputs, @Output, eventEmmitter, @ViewChild in Angular 2?
   1. @Input decorator to accept an input binding, we can do the same and listen in the parent for when a value changes inside our child component.we can import Output and decorate a new change property inside our Component. we need to import and bind a new instance of the EventEmitter.
   2. @Inputs - @Input allows you to pass data into your controller and templates through html and defining custom properties. This allows you to easily reuse components and have them display different values for each instance of the renderer.
   3. @Output - Components push out events using a combination of an @Output and an EventEmitter. This allows a clean separation between reusable Components and application logic.
   4. eventEmmitter -
   5. @Component({...})
   6. export class CounterComponent {
   7. @Input()
   8. count: number = 0;
   9. @Output()
   10. change: EventEmitter<number> = new EventEmitter<number>();
   11. increment() {
   12. this.count++;
   13. this.change.emit(this.count);
   14. }
   15. decrement() {
   16. this.count--;
   17. this.change.emit(this.count);
   18. }
   19. }
   20. @view Child -
   21. import {Component, ViewChild} from '@angular/core';
   22. import {ChildComponent} from './child.component';
   23. @Component({
   24. selector: 'my-app',
   25. template: `
   26. <div>
   27. <h1>Parent Component</h1>
   28. <button (click)="showHideText()">Show/Hide Child Component Text</button>
   29. <child-component></child-component>
   30. </div>
   31. `,
   32. directives: [ChildComponent]
   33. })
   34. export class AppComponent {
   35. @ViewChild(ChildComponent) private childComponent:ChildComponent;
   36. showHideText() {
   37. this.childComponent.toggleVisibility('Parent Component');
   38. }
   39. }
2. What are the immutable variables and reference variable?

1. var, let, const (immutable)

2. object, array, function (reference)

1. How many directives in angular?

1. Attribute Directive - NgStyle, NgClass

2. Structural Directive - NgIf, NgFor, NgSwtich,

1. Cold Observables vs Hot Observables?

- Cold Observables

const obsv = new Observable(observer => {

setTimeout(() => {

observer.next(1);

}, 1000);

setTimeout(() => {

observer.next(2);

}, 2000);

setTimeout(() => {

observer.next(3);

}, 3000);

setTimeout(() => {

observer.next(4);

}, 4000);

});

// Subscription A

setTimeout(() => {

obsv.subscribe(value => console.log(value));

}, 0);

// Subscription B

setTimeout(() => {

obsv.subscribe(value => console.log(`>>>> ${value}`));

}, 2500);

out put-

"A B"

1

2

3

" 1"

4

" 2"

" 3"

" 4"

- Hot Observables

const obsv = new Observable(observer => {

setTimeout(() => {

observer.next(1);

}, 1000);

setTimeout(() => {

observer.next(2);

}, 2000);

setTimeout(() => {

observer.next(3);

}, 3000);

setTimeout(() => {

observer.next(4);

}, 4000);

}).publish();

hotObsv.connect();

// Subscription A

setTimeout(() => {

obsv.subscribe(value => console.log(value));

}, 0);

// Subscription B

setTimeout(() => {

obsv.subscribe(value => console.log(` ${value}`));

}, 2500);

out put-

"A B"

1

2

3

" 3"

4

" 4"

1. What is JiT Compiler and AoT compiler?

The JiT (Just-in-time) compiler compiles the application dynamically, as the application loads. To do this, we will need to rely on 3 providers that tell the JiT compiler how to translate the template texts for a particular language:

-TRANSLATIONS is a string containing the content of the translation file.

-TRANSLATIONS\_FORMAT is the format of the file.

-LOCALE\_ID is the locale of the target language.

To Internationalize with the AoT (Ahead of time) compiler, you will have to:

-pre-build a seperate application package for each language

-determine which language the user needs

-serve the appropriate application package

1. Explain Call, Bind, Apply, Prototype? with example.

----

Call

var person1 = {firstName: 'Jon', lastName: 'Kuperman'};

var person2 = {firstName: 'Kelly', lastName: 'King'};

function say(greeting) {

console.log(greeting + ' ' + this.firstName + ' ' + this.lastName);

}

say.call(person1, 'Hello'); // Hello Jon Kuperman

say.call(person2, 'Hello'); // Hello Kelly King

bind

var person1 = {firstName: 'Jon', lastName: 'Kuperman'};

var person2 = {firstName: 'Kelly', lastName: 'King'};

function say() {

console.log('Hello ' + this.firstName + ' ' + this.lastName);

}

var sayHelloJon = say.bind(person1);

var sayHelloKelly = say.bind(person2);

sayHelloJon(); // Hello Jon Kuperman

sayHelloKelly(); // Hello Kelly King

Apply

var person1 = {firstName: 'Jon', lastName: 'Kuperman'};

var person2 = {firstName: 'Kelly', lastName: 'King'};

function say(greeting) {

console.log(greeting + ' ' + this.firstName + ' ' + this.lastName);

}

say.apply(person1, ['Hello']); // Hello Jon Kuperman

say.apply(person2, ['Hello']); // Hello Kelly King

Prototype  
function Person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}

var myFather = new Person("John", "Doe", 50, "blue");  
var myMother = new Person("Sally", "Rally", 48, "green");

1. What is use of async and sync.

**Synchronous way:** It waits for each operation to complete, after that only it executes the next operation

**Asynchronous way:** It never waits for each operation to complete, rather it executes all operations in the first GO only. The result of each operation will be handled once the result is available