

Q1 is mapped to CLO 2
CLO 2
Question 01: Applying the concepts of JavaScript, provide the output of the following programs. (1*10 = 10 marks)

✓ 1. `var num = 8;
var num = 10;
console.log (num);`

2. `function sayHi() {
 console.log (name);
 var name = 'Harris';
}
sayHi();`

✓ 3. `let number = 0;
console.log (number++);
console.log (++number);
console.log (number);`

✓ 4. `function sum (a, b) {
 return a + b;
}
console.log (sum(1, '2'));`

✓ 5. `function sayHi() {
 return (() => 0)();
}
console.log (typeof sayHi());`

✓ 6. `console.log ([...'Sarmad']);`

✓ 7. `function sum (a, b) {
 return arguments.reduce((a, b) => a + b);
}
console.log (sum (1,2,3));`

✓ 8. `console.log (1 == '1');
console.log (false == '0');
console.log (true == '1');
console.log ('1' == '01');`

✓ 9. `const obj = {a: 'one', b: 'two', a: 'three'};
console.log (obj);`

✓ 10. `function getAge(...args) {
 console.log (typeof args);
}
getAge(21)`

main.js	Run	Output
<pre> 1 var num = 8 2 var num = 10 3 console.log(num) </pre>		<pre> node /tmp/onJHL50410.js 10 </pre>

main.js	Run	Output
<pre> 1 function sayHi(){ 2 console.log(name) 3 var name = "Alliyan" 4 } 5 sayHi() </pre>		<pre> node /tmp/onJHL50410.js undefined </pre>

main.js	Run	Output
<pre> 1 let number = 0 2 console.log(number++) 3 console.log(++number) 4 console.log(number) </pre>		<pre> node /tmp/onJHL50410.js 0 2 2 </pre>

main.js	Run	Output
<pre> 1 function sum (a, b){ 2 return a+b 3 } 4 console.log(sum(1, '2')) </pre>		<pre> node /tmp/onJHL50410.js 12 </pre>

main.js	Run	Output
<pre> 1 function sayHi(){ 2 return (() => 0)(); 3 } 4 console.log(typeof sayHi()) </pre>		<pre> node /tmp/onJHL50410.js number </pre>

main.js	Run	Output
<pre> 1 console.log([... 'Sammad']) </pre>		<pre> node /tmp/onJHL50410.js ['S', 'a', 'm', 'm', 'a', 'd'] </pre>

main.js	<div><div>Run</div></div>	Output
<pre>1- function sum (a,b) { 2 return arguments.reduce((a,b) => a+b) 3 } 4 console.log(sum(1,2,3))</pre>		<pre>node /tmp/onJHL50410.js /tmp/onJHL50410.js:2 return arguments.reduce((a,b) => a+b) ^ TypeError: arguments.reduce is not a function at sum (/tmp/onJHL50410.js:2:22) at Object.<anonymous> (/tmp/onJHL50410.js:4:13) at Module._compile (internal/modules/cjs/loader.js:999:30) at Object.Module._extensions..js (internal/modules/cjs/loader.js:1027:10)</pre>

main.js	<div><div>Run</div></div>	Output
<pre>1- // function sum (a,b){ 2 // return arguments.reduce((a,b) => a+b) 3 // } 4 // console.log(sum(1,2,3)) 5 6 console.log(1=='1') 7 console.log(false=='0') 8 console.log(true=='1') 9 console.log('1'=='01')</pre>		<pre>node /tmp/onJHL50410.js true true true false</pre>

main.js	<div><div>Run</div></div>	Output
<pre>1 const obj = {a: 'one', b: 'two', a: 'three'} 2 console.log(obj)</pre>		<pre>node /tmp/onJHL50410.js { a: 'three', b: 'two' }</pre>

main.js	<div><div>Run</div></div>	Output
<pre>1- function getAge(...args){ 2 console.log(typeof args) 3 } 4 getAge(21)</pre>		<pre>node /tmp/onJHL50410.js object</pre>

Question 2

What is Redux?

Redux is a standalone state management library, which can be used with any library or framework. If your background is **React developer**, then you have used the **Redux library with React**. The primary use of **Redux** is that we can use one application state as a global state and interact with the state from any react component is very easy whether they are siblings or parent-child.



Redux Logo

Basic usage of Redux comes into picture when the app gets large and complex. In such apps, simple data management as parent-child becomes difficult using `props`. There are multiple components trying to communicate with multiple other components. In such cases, Redux comes in handy.

1.

- `renderItem`
- `data`

1.

The Push action adds a route on top of the stack and navigates forward to it. This differs from navigate in that navigate will pop back to earlier in the stack if a component is already mounted there. Push will always add on top, so a component can be mounted multiple times.

1.

AsyncStorage:

AsyncStorage is not secure for sensitive data. AsyncStorage simply saves data to documents on the phone's hard drive, and therefore anyone with access to the phone's file system can read that data. Of course, whether or not this is problematic for you depends on what you mean by "sensitive data." It stores the data in unencrypted form.

Async Storage is great but it **lacks security**. This is less than ideal when storing sensitive data such as access tokens, payment information and so on. One limitation of an AsyncStorage API is that **on Android the size of the database is set to a default of 6MB limit**. Also, AsyncStorage storage is based on SQLite. Thus, it's important to keep SQLite limitations in mind, too.

Expo Secure Store:

If you're using Expo you can use Expo.SecureStore to encrypt and securely store key–value pairs locally on the device.

Android: Values are stored in [SharedPreferences](#), encrypted with [Android's Keystore system](#).

Size limit for a value is 2048 bytes.

1.

Initial params

You can also pass some initial params to a screen. If you didn't specify any params when navigating to this screen, the initial params will be used. They are also shallow merged with any params that you pass. Initial params can be specified with an

`initialParams` prop:

```
<Stack.Screen
  name="Details"
  component={DetailsScreen}
  initialParams={{ itemId: 42 }}
/>
```

1.

What is useEffect in react native?



What does useEffect do? By using this Hook, **you tell React that your component needs to do something after render**. React will remember the function you passed (we'll refer to it as our “effect”), and call it later after performing the DOM updates.

1. No dependency passed:

```
useEffect(() => {  
  //Runs on every render  
});
```

2. An empty array:

```
useEffect(() => {  
  //Runs only on the first render  
}, []);
```

3. Props or state values:

```
useEffect(() => {  
  //Runs on the first render  
  //And any time any dependency value changes  
}, [prop, state]);
```

1.

animationType

The `animationType` prop controls how the modal animates.

Possible values:

- `slide` slides in from the bottom,
- `fade` fades into view,
- `none` appears without an animation.

TYPE	DEFAULT
<code>enum('none', 'slide', 'fade')</code>	<code>none</code>

1.

onLongPress

Called if the time after `onPressIn` lasts longer than 500 milliseconds. This time period can be customized with `delayLongPress`.

TYPE
<code>((nativeEvent: PressEvent)) => void</code>

onPress

Called after `onPressOut`.

TYPE
<code>((nativeEvent: PressEvent)) => void</code>

onPressIn

Called immediately when a touch is engaged, before `onPressOut` and `onPress`.

TYPE
<code>((nativeEvent: PressEvent)) => void</code>

onPressOut

Called when a touch is released.

TYPE
<code>((nativeEvent: PressEvent)) => void</code>

pressRetentionOffset

Additional distance outside of this view in which a touch is considered a press before `onPressOut` is triggered.

TYPE	DEFAULT
<code>Rect or number</code>	<code>{bottom: 30, left: 20, right: 20, top: 20}</code>

1.

ScrollView will mount all the children at once while FlatList will calculate the items that are way off of distance based on the scroll and will unmount them. Hence the state is lost and re-created.

The key difference between ScrollView vs FlatList is that ScrollView will load all those items as the page appears or the component is loaded. Hence, if you have thousands of items to load, it would make your app very slow and inefficient. On the other side, FlatList, will load 10 by default and as the user scrolls the next ones will show up on the screen.

ScrollView will load the items (data for scrolling) immediately after the component has been loaded. As a result, all data will be stored in **RAM**, and you will be unable to use hundreds or thousands of items in it (*due to low performance*).

FlatList, on the other hand, has a better solution for this problem; it will mount 10 items (by default) to the screen, and as the user scrolls the view, other items will mount. It's a significant advantage to use FlatList instead of ScrollView.