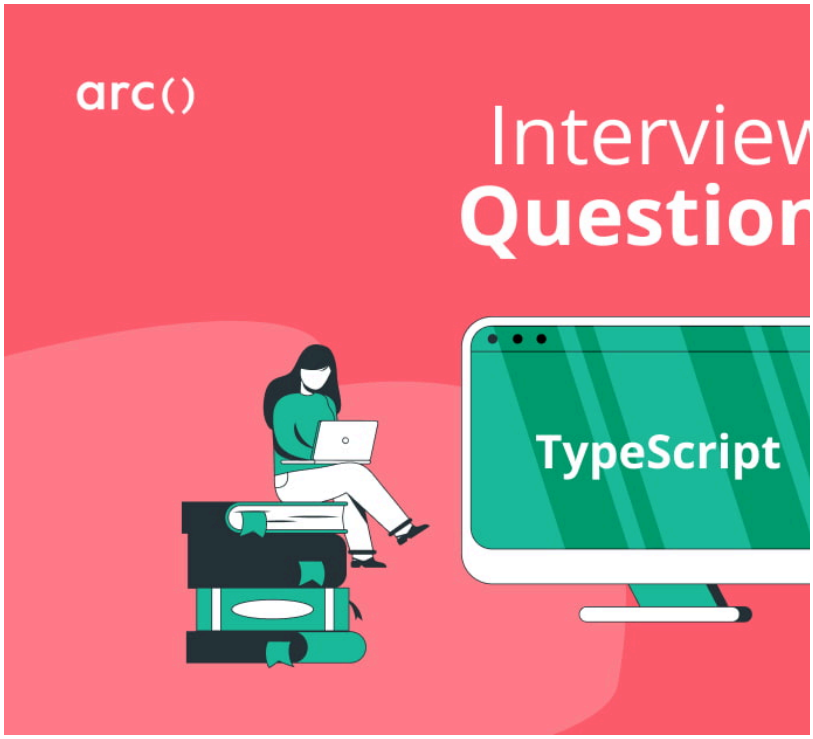
arc()

Interview Preparation

# 20 TypeScript Intervie and Answers You Sho For

2025-11-24 27 min read Written by V




Written by
**William Juan**

**Summary: Use this list of TypeScript interv answers to prepare for your upcoming mee recruiter or lead engineer!**

Looking to hire a TypeScript developer or hop TypeScript developer? These **TypeScript inte answers** can help you put together a comprel

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

specifically about JavaScript.

In this guide, we break down the important Ty
to know into three groups:`

- Basic TypeScript Interview Questions
- Intermediate TypeScript Interview Questions
- Advanced TypeScript Interview Questions

You can expect to learn about what questions
answers to expect, and what to focus on evalu
Additionally, we'll explain *why you should ask
look for, and what you want to draw out from y*

And, at the end of the article (as well as a few
link you to some other helpful interview advice

Let's get started!

Still manually sourcing candidates?
Meet HireAI
Sta

**Related Guides**: JavaScript Interview Q
Interview Questions • React Interview Qu
Interview Questions • Java Interview Qu

arc()

Menu

Finding Opportunities
Application Prep
Interviews
Getting Hired
Career Growth
Remote Work
More
Find Jobs

# Basic TypeScript Interview

arc()

## 1. What is TypeScript and how JavaScript?

*This is a fundamental question that interviewe candidate's familiarity with the language and h with its features. Make sure your candidate ca language, its core features, and how it relates*

TypeScript is a **superset** of JavaScript that co Conceptually, the relationship between TypeS comparable to that of SASS and CSS. In othe JavaScript's ES6 version with some additiona

TypeScript is an **object-oriented** and **statical** to Java and C#, whereas JavaScript is a scrip Python. The object-oriented nature of TypeSc such as classes and interfaces, and its static t tooling with type inference at your disposal.

From a code perspective, TypeScript is writter a `.ts` extension whereas JavaScript is writter Unlike JavaScript, TypeScript code is not und and can't be executed directly in the browser The `.ts` files need to be transpiled using Typ plain JavaScript first, which then gets execute

## 2. What are the benefits of usi

*Each programming language has its pros and behind its applications. You might ask this que*

arc()

Menu

Finding Opportunities
Application Prep
Interviews
Getting Hired
Career Growth
Remote Work
More
Find Jobs

arc()

*Instead of just listing out TypeScript's benefits your candidates' to talk about how a typical pr TypeScript. This could range from better main increased developer productivity because of t offers.*

An immediate advantage of using TypeScript strongly typed language that uses type inferer open the doors to better tooling and tighter int TypeScript's strict checks catch your errors ea chances of typos and other human errors from production. From an IDE's perspective, TypeS opportunity for your IDE to understand your c display better hints, warnings, and errors to th TypeScript's **strict null check** throws an erro IDE) preventing a common JavaScript error o property of an undefined variable at runtime.

A long-run advantage of using TypeScript is it maintainability. The ability to describe the sha directly in your code makes your codebase ea predictable. When used correctly, TypeScript standardized language resulting in better reac time and effort down the road as the codebas

**Read More**: Common Non-Technical Int Software Developer Jobs

## 3. What are interfaces in TypeS

arc()

*feature. An ideal answer should also include i*
*using interfaces.*

Interfaces are TypeScript's way of defining the
words, interfaces are a way to describe data s
array of objects.

We declare interfaces with the help of the `int`
by the interface name and its definition. Let's
a user object

```
interface User {
  name: string;
  age: number;
}
```

The interface can then be used to set the type
you assign primitive types to a variable). A var
then conform to the interface's properties.

```
let user: User = {
  name: "Bob",
  age: 20, // omitting the `age` proper
different type instead of a number woul
};
```

Interfaces help drive consistency in your Type
interfaces also improve your project's tooling,
autocomplete functionality in your IDEs and e
are being passed into constructors and functio

arc()

arc()

*Being able to modify an interface is useful to r*
*max_Being able to modify an interface is usef*
*and maximize the reusability of existing interfa*
*This question covers one of the many features*
*create a new interface from an existing interfa*

TypeScript has a utility type called `omit` that l
by passing a current type/interface and select
from the new type. The example below shows
type `UserPreview` based on the `User` interf
the `email` property.

```
interface User {
  name: string;
  description: string;
  age: number;
  email: string;
}

// removes the `email` property from th
type UserPreview = Omit<User, "email">;

const userPreview: UserPreview = {
  name: "Bob",
  description: "Awesome guy",
  age: 20,
};
```

**Read More**: 31 Questions to Ask at an I
Development Jobs

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

*Enums are a common data structure in most t*
*Understanding enums and how to use them is*
*code and making code more readable. The ca*
*explain enums at a high level, along with their*

Enums or enumerated types are a means of c
constants. These data structures have a cons
of constant values. Enums in TypeScript are c
a set number of options for a given value usin

Let's look at an example of an enum to define

```
enum UserType {
  Guest = "G",
  Verified = "V",
  Admin = "A",
}

const userType: UserType = UserType.Ver
```

Under the hood, TypeScript translates enums
after compilation. This makes the use of enum
to using multiple independent `const` variables
offer makes your code type-safe and more rea

## 6. What are arrow functions in

*Arrow function is a popular feature of ES6 and*
*an alternate shorter way of defining functions.*
*functions come with their pros and cons, whic*
*when choosing which method to use.*

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

create callback functions in TypeScript. Array
as `map`, `filter`, and `reduce` all accept arro
arguments.

However, arrow functions' anonymity also has
the shorter arrow function syntax can be more
Furthermore, arrow functions' nameless natur
create self-referencing functions (i.e. recursio

Let's take a look at a regular function that acc
returns its sum.

```
function addNumbers(x: number, y: numbe
  return x + y;
}

addNumbers(1, 2); // returns 3
```

Now let's convert the function above into an a

```
const addNumbers = (x: number, y: numbe
  return x + y;
};

addNumbers(1, 2); // returns 3
```

**Read More**: Time Management Skills fo
Tips, Tools, and Strategies

## 7. What are the differences bet
and `const`?

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

*keywords and understanding which to use wh*
*writing quality code. As the candidate, make s*
*case for each of the keywords along with thei*

• **var**: Declares a function-scoped or global va
and scoping rules to JavaScript's `var` variable
require setting its value during its declaration.

• **let**: Declares a block-scoped local variable.
setting a value of a variable during its declarat
variable means that the variable can only be a
containing block, such as a function, an if/else
Furthermore, unlike `var`, `let` variables cannc
before they are declared.

```
// reading/writing before a `let` varia
console.log("age", age); // Compiler Er
Block-scoped variable 'age' used before
age = 20; // Compiler Error: error TS24
variable 'age' used before its declarat

let age: number;

// accessing `let` variable outside it'
function user(): void {
  let name: string;
  if (true) {
    let email: string;
    console.log(name); // OK
    console.log(email); // OK
  }
  console.log(name); // OK
  console.log(email); // Compiler Error
```

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

arc()

• **const**: Declares a block-scoped constant va
after it's initialized. `const` variables require ar
declaration. This is ideal for variables that dor
lifetime.

```
// reassigning a `const` variable
const age: number = 20;
age = 30; // Compiler Error: Cannot ass
is a constant or read-only property

// declaring a `const` variable without
const name: string; // Compiler Error:
be initialized
```

***Sidenote**: Linters does a great job in catching
compilers will throw an error if a rule is violate
these keywords.*

## 8. When do you use a return ty
## how does it differ from `void`?

*This is a question that showcases the candida
TypeScript's types and their use cases in the
types. The candidate should be able to disting
and identify the return type of a function by loo*

Before diving into the differences between ne
about the behavior of a JavaScript function wh
explicitly.

Let's take the function in the example below. I
anything to the caller. However, if you assign i

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

```
printName(name: string): void {
  console.log(name);
}

const printer = printName('Will');
console.log(printer); // logs "undefine
```

The above snippet is an example of `void` fun
explicit returns are inferred by TypeScript to h

In contrast, `never` is a type that represents a
example, a function with an infinite loop or a fu
are functions that have a `never` return type.

```
const error = (): never => {
  throw new Error("");
};
```

In summary, `void` is used whenever a functio
explicitly whereas `never` is used whenever a

> **Read More**: Problem-Solving Skills for S
> Why & How to Improve

## 9. What access modifiers are s TypeScript?

*Access modifiers and encapsulation go hand-
is an opportunity to understand the candidate
of encapsulation and TypeScript's approach to*

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

The concept of "**encapsulation**" is used in ob
to control the visibility of its properties and me
access modifiers to set the visibility of a class
TypeScript gets compiled to JavaScript, logic
is applied during compile time, not at run time

There are three types of access modifiers in
TypeScript: `public`, `private`, and `protect`

• **public**: All properties and methods are publi
of a class are visible and accessible from any

• **protected**: Protected properties are accessil
class and its subclass. For example, a variabl
the `protected` keyword will be accessible fr
and within a different class that extends the cl
or method.

• **private**: Private properties are only accessib
property or method is defined.

To use any of these access modifiers, add the
or `public` (if omitted, TypeScript will default t
property or method.

```
class User {
  private username; // only accessible

  // only accessible inside the `User`
  protected updateUser(): void {}

  // accessible from any location
```

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

Violating the rules of the access modifier, such
class's private property from a different class
error during the compilation process.

```
Property '<property-name>' is private a
within class '<class-name>'.
```

In conclusion, access modifiers play an impor
code. They allow you to expose a set of public
implementation details. You can think of acces
entry gates to your class. Effective use of acc
chance of errors from misusing another class'

## 10. What are generics and how
## TypeScript?

*In addition to the "what", an interviewer asking
looking for answers on how generics tie into th
reasoning behind using generics over other ty
be comfortable discussing what generics are,
benefits.*

Good software engineering practice often enc
flexibility. The use of **generics** provides reusa
allowing a component to work over a variety o
one while preserving its precision (unlike the u

Below is an example of a generic function tha
type to be used within the function.

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

```
  return arg;
}
```

To call a generic function, you can either pass
angle brackets or via *type argument inference*
type based on the type of the argument passe

```
// explicitly specifying the type
let user = updateUser<string>("Bob");

// type argument inference
let user = updateUser("Bob");
```

Generics allows us to keep track of the type ir
function. This makes the code flexible and reu
on its type accuracy.

*Looking to hire the best remote developers? A*

- ⚡ *Get instant candidate matches without sea*
- ⚡ *Identify top applicants from our network of*
- ⚡ *Hire 4x faster with vetted candidates (quali*

***Try Arc to hire top developers now →***

# Intermediate TypeScript Int Questions

The following set of questions should test the
knowledge of TypeScript and some of its wide

## 1. When should you use the ur

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

*preferred option because of its safer typing. Y*
*to explain what the unknown type is used for*

unknown is a special type that is similar to an
case of the unknown type is when you don't k
upfront. unknown variables accept any value.
operate on an unknown variable, TypeScript r
type assertion. This difference makes unknow

Let's look at two examples to highlight the diff
between any and unknown types.

The snippet below shows a valid TypeScript c
any value as the `callback` parameter, and
the `invokeCallback` method will try to call i
any compile-time error, however, it could lead
pass in a variable that's not callable as the ca

```
invokeCallback(callback: any): void {
  callback();
}
```

The unknown equivalent of the above examp
check before calling the `callback` function. T
prevents the run time error above from happe

```
invokeCallback(callback: unknown): void
  if (typeof callback === 'function') {
    callback();
  }
}
```

arc()

assertion to perform any operation on the vari

## 2. What is `noImplicitAny` and purpose?

*This is a TypeScript configuration question tha the level of familiarity the candidate has when project. Although this is only one property fron configurable properties, `noImplicitAny` pla projects.*

`noImplicitAny` is a property in a TypeScrip (`tsconfig.json`) that modifies how TypeSc project's implicit any types. The `noImplicit` true or false and can always be changed later as to what this value should be as each projec decide on what the flag should be set to, it is i differences between when it's turned on and v

When the `noImplicitAny` flag is false (by d infer the variable type based on how it's used. defaults the type to any.

On the other hand, when the `noImplicitAn` will attempt to infer the type, throwing a compi isn't able to infer the type.

> **Setting the `noImplicitAny` flag to either tru you from setting a variable's type to any.**

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

arc()

Your Developer Career)

## 3. What are conditional types i

*Sometimes, a variable's type is dependent on help bridge the input and output's type – trans based on a set condition. Conditional type is a feature that helps write type-safe framework-l boundaries. Although the candidate might not every day, they might be using them indirectly frameworks or libraries.*

Conditional types in TypeScript are similar to t name suggests, it assigns a type to the variab

The general expression for defining a conditio follows:

```
A extends B ? X : Y
```

Type X in the snippet above is applied if the va whereas type Y is applied if the variable doesn type X is assigned if A extends B, while type Y extend B.

## 4. What is the difference betwe intersection types?

*Union and intersection types are considered a TypeScript. Understanding this is important w developer's TypeScript projects to reduce dup*

arc()

arc()

Unions and intersection types let you compos
types instead of creating them from scratch. B
come with their unique characteristics which r
use cases.

A **union** type is described as a type that can b
Union type uses the | (vertical bar) symbol to
that will be used in the new type. Let's take a

```
interface B {
  name: string,
  email: string
}

interface C {
  name: string,
  age: number
}

type A = B | C;
```

A in the snippet above can either be of type B
interfaces, A can either contain `name` and `ema`
but not `email` and `age`.

When accessing type A, TypeScript only lets y
exist in both B and C (`name`) as those are the
can be certain of their existence.

**Intersection** on the other hand, is described a
multiple types into one – combining all the pro
create a new type. Intersection uses the & syr
types that will be combined. Let's look at an e

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

```
    name: string,
    email: string
}

interface C {
    name: string,
    age: number
}

type A = B & C;
```

A in the snippet above will contain all the prop
both B and C (`name`, `email`, and `age`).

> **Read More**: Interpersonal Skills: What D
> Know (& How to Improve)

## 5. What is the difference betwe `implements` in TypeScript?

*Inheritance is an important concept in object-c specifically for polymorphism and code reusal keywords that TypeScript uses for inheritance `extends` and `implements`. This question is candidate's knowledge of the concept of inher TypeScript.*

Implements and extends have different uses.
individual and take a look at an example to co

When a class **extends** another class, the chil properties and methods of the class it extends the existing properties and methods of its pare

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

interface. The `implements` keyword acts as
to follow, and TypeScript will make sure that th
shape as the class or interface it implements.

Let's look at an example of a class and how a
implement it.

```
class User {
  name: string;
  age: number;
}

// John will contain name and age from
class John extends User {}

// this will result in an error as Bob
properties that
// the User class has
class Bob implements User {}

// This is valid as Mike satisfies the
the
// User class
class Mike implements User {
  name = 'Mike';
  age = 25
}
```

Still manually sourcing candidates?
Meet HireAI
Sta

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

arc()

The following set of advanced questions for T
test your candidates' deep knowledge of TS a
level concepts and features.

## 1. Explain how optional chaini
## TypeScript.

*Optional chaining is a TypeScript feature that
accessing of values inside deep objects. Unde
use this feature will help developers write mor
makes sense). A candidate should be able to
chaining is and also elaborate on its use case*

Optional chaining allows you to access proper
deep within an object without having to check
reference in the chain.

Optional chaining uses the question mark follo
operator. TypeScript evaluates each reference
null or undefined check before accessing its c
immediately stops the execution when it fails t
and returns `undefined` for the entire chain.

The code snippet below is an example of acce
without using optional chaining.

```
const user = {
  personalInfo: {
    name: 'John'
  }
}

// without optional chaining
const name = user && user.personalInfo
```

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

```
// with optional chaining
const name = user?.personalInfo?.name;
```

## 2. What are abstract classes?

*Abstract class is another feature of TypeScrip*
*concept in object-oriented programming (OOF*
*to get an insight into the candidate's understa*
*advanced features in TypeScript.*

Abstract classes specify a contract for the obj
instantiate them directly. However, an abstrac
implementation details for its members.

An abstract class contains one or more abstra
that extend the abstract class will then have to
for the superclass's abstract members.

Let's look at an example of how an abstract cl
and how another class can extend it. In the ex
both `Car` and `Bike` extend the `Vehicle` clas
a different implementation of the `drive()` me

```
abstract class Vehicle {

  abstract drive(): void;

  startEngine(): void {
    console.log('Engine starting...');
  }
}

class Car extends Vehicle {
  drive(): void {
```

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

```
}

class Bike extends Vehicle {
  drive(): void {
    console.log('Driving on a bike');
  }
}
```

**Read More**: 8 Great Benefits of Working for Remote Developers)

## 3. What are type assertions in

*Type assertions are a way to tell TypeScript th types are widely used in TypeScript applicatio scenarios where you want to specify a more s candidate should be able to explain not only w also their syntax and the reasons behind why*

Type assertion allows you to explicitly set the compiler not to infer it. This is useful when you more specifically than its current type or curre cases, you can use type assertions to tell Typ the variable.

TypeScript provides two syntaxes for type ass

```
// using the `as` keyword
const name: string = person.name as str

// using `<>`
const name: string = <string>person.nam
```

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

TypeScript projects.

## 4. What is the difference betwe and contextual typing?

*Type inference is a key feature in TypeScript t code that is both less error-prone and easier t question to gauge your candidate's understan type inference and their overall familiarity with*

TypeScript can infer the type of a variable usu initialization or declaration. This process is kn

The snippet below is an example of type infer the type of `name` as "string" based on the valu

```
let name = 'john'
```

**Contextual typing** is a subset of type inferen the location or context of a variable to infer its this as type inference in the opposite direction

In the snippet below, TpyeScript uses informa the `onmousedown` function to infer the type o the `mouseEvent` parameter.

```
window.onmousedown = function(mouseEver
   console.log(mouseEvent.button);   //
   console.log(mouseEvent.person);   //
};
```

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

arc()

Developer

## 5. How does function overload TypeScript?

*Function overloading helps make functions m allowing the same function to behave a differe input passed in. In addition to the how, you m on how function overloads tie into the code re other benefits it provides.*

Function overload is when the same function with a different set of arguments – the numbe return types.

Let's look at an example of how a `print` func types as its parameter by using function overl

```
print(message: string): void;
print(message: string[]): void;

print(message: unknown): void {
  if (typeof message === 'string') {
    console.log(message);
  } else if (Array.isArray(message)) {
    message.forEach((individualMessage)
      console.log(individualMessage);
    });
  } else {
    throw new Error('unable to print');
  }
}
```

arc()

Menu

Finding Opportunities

Application Prep

Interviews

Getting Hired

Career Growth

Remote Work

More

Find Jobs

```
print('Single message');
// Console Output:
// Single message

print(['First message', 'Second message
// Console Output
// First message
// Second message
```

Apart from the reusability of the function, func
with autocomplete support. When calling a fur
IDE), you will be provided with a list of all poss
choose from for your specific use case, creati
experience.

## Conclusion

Whether you're a hiring manager looking for tl
developer preparing for an interview, we hope
questions help you through the process.

Keep in mind that technical skills and knowlec
hiring process. Past experiences and soft skill
make sure you hire the best candidate (or lan

Good luck on your upcoming TypeScript interv
helpful guides to read:

- 8 Common Interview Mistakes Remote Softv
- 8 Behavioral Interview Questions Asked by T
- 10+ Tips for Preparing for a Remote Softwar

arc()

arc()

- Phone Screen Interview vs Actual Phone Int
Differences
- How to Write a Great Thank-You Email After

*You can also explore HireAI to skip the line ar*

⚡ *Get instant candidate matches without sea*
⚡ *Identify top applicants from our network of*
*screening*
⚡ *Hire 4x faster with vetted candidates (quali*

***Try HireAI and hire top developers now →***

Share:

Written by
**William Juan**

**Web & Mobile Front-End Dev**

William is a front-end developer working primarily in the w
majority of his work has revolved around the Angular ecos
Angular-related frameworks such as NativeScript and Ionic. A
gained over years as a writer on software de

## Further readi

arc()

**Menu**

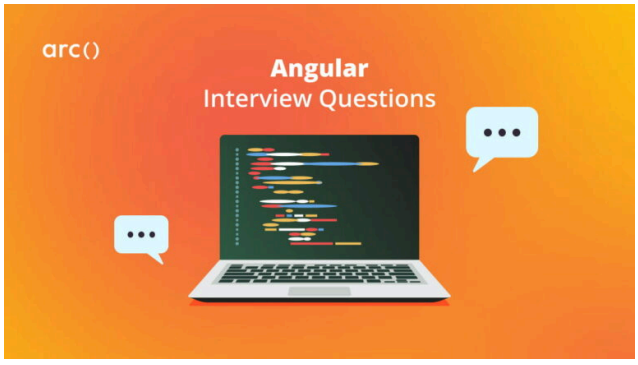
Finding Opportunities
Application Prep
Interviews
Getting Hired
Career Growth
Remote Work
More
Find Jobs

arc()

Interview Preparation

## 20 Spring Interview Questions and Answers to Know (With MVC & Boot)

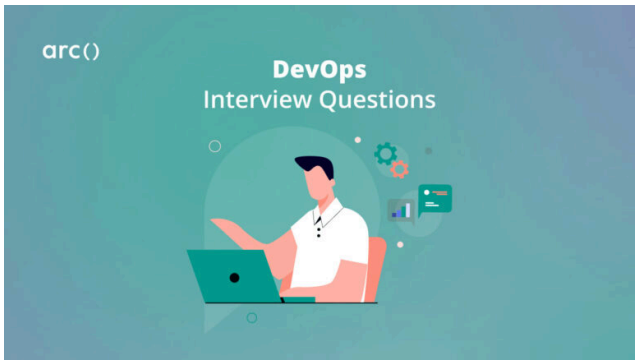
2024-11-28




Interview Preparation

## 29 Angular Interview Questions and Answers to Practice & Prepare For

2024-04-17




Interview Preparation

arc()

Menu

- Finding Opportunities
- Application Prep
- Interviews
- Getting Hired
- Career Growth
- Remote Work
- More
- Find Jobs

arc()

# and Answers to Know

2024-04-17

# to Pr Adv)

2024-04-17


BRANDS

**Arc**
Helping employers find developers for remote jobs

**Codementor**
Find a mentor to help you in real time

LINKS

Hire Freel
Hire Freel
Hire Freel
Hire Freel
Hire Freel
Hire Freel
Remote D
Freelance
FAQs Abo
About Us

JOBS BY EXPERTISE

Remote Jobs
Remote Developer Jobs
Remote Design Jobs
Remote Marketing Jobs
Remote Front-End Developer Jobs
Remote Back-End Developer Jobs
Remote DevOps Engineer Jobs
Remote Data Scientist Jobs
Remote Data Engineer Jobs
Remote Mobile App Developer Jobs
Remote Game Developer Jobs

JOBS BY

Remote Ja
Remote A
Remote C
Remote Ja
Remote P
Remote R
Remote S
Remote .N
Remote A
Remote G

arc()

Menu

Finding Opportunities
Application Prep
Interviews
Getting Hired
Career Growth
Remote Work
More
Find Jobs

© Copyright 2026 Arc · All rights reserved · Cookie Policy · P