Here is a comprehensive set of 100 advanced and hard-level multiple-choice questions (MCQs) on the specified TypeScript topics. These questions cover a wide range of TypeScript concepts and scenarios to challenge your understanding.

HelloWorld

1. **What is the output of the following TypeScript code?**

   ```typescript
   console.log("Hello, World!");
   ```

   - A. `Hello, World!`
   - B. `undefined`
   - C. `null`
   - D. `Error: console.log is not a function`

2. **How do you compile a TypeScript file named `helloworld.ts`?**

   - A. `ts-node helloworld.ts`
   - B. `node helloworld.ts`
   - C. `tsc helloworld.ts`
   - D. `npm run helloworld.ts`

JSON Objects

3. **What is the correct way to define a JSON object in TypeScript?**

   ```typescript
   const person = {
     name: "John",
     age: 30
   };
   ```

```

- A. `const person: JSON`
- B. `const person: object`
- C. `const person: { name: string; age: number }`
- D. `const person: any`

4. **How can you parse a JSON string into a TypeScript object?**

```typescript
const jsonString = '{"name": "John", "age": 30}';
```

- A. `const obj = JSON.parse(jsonString);`
- B. `const obj: { name: string; age: number } = JSON.parse(jsonString);`
- C. `const obj = jsonString;`
- D. `const obj = JSON.stringify(jsonString);`

Syntax Error

5. **Identify the syntax error in the following TypeScript code:**

```typescript
let num: number = 5
```

- A. Missing type annotation
- B. Missing semicolon
- C. Incorrect type assignment
- D. Variable not initialized

6. **What is the result of running TypeScript code with a syntax error?**

- A. The code runs with warnings
- B. The code fails to compile
```

- C. The code runs but skips the erroneous part
- D. The code corrects the error automatically

Type Error

7. **What type of error will this TypeScript code produce?**

```typescript
let str: string = 5;
```

- A. Syntax error
- B. Type error
- C. Runtime error
- D. No error

8. **Which TypeScript feature helps in identifying type errors at compile time?**

- A. Type Inference
- B. Type Annotations
- C. Type Guards
- D. Type Aliases

Assignability Error

9. **Which of the following will produce an assignability error?**

```typescript
let x: number = "Hello";
```

- A. `let x: string = 10;`
- B. `let x: number = "Hello";`
- C. `let x: boolean = true;`

- D. `let x: any = "Hello";`

10. **How can you prevent assignability errors in TypeScript?**
    - A. Use `any` type for all variables
    - B. Avoid using type annotations
    - C. Ensure variables are assigned values of the correct type
    - D. Disable type checking

Strong Typing

11. **Which of the following statements about strong typing in TypeScript is correct?**
    - A. Strong typing means variables can change types at runtime.
    - B. Strong typing ensures type errors are caught at compile time.
    - C. Strong typing allows for dynamic type assignment.
    - D. Strong typing is not supported in TypeScript.

12. **How does TypeScript enforce strong typing?**
    - A. Through runtime type checks
    - B. Through compile-time type checks
    - C. By disabling type inference
    - D. By using the `any` type

const and let

13. **What is the key difference between `const` and `let` in TypeScript?**
    - A. `const` variables can be reassigned, `let` variables cannot.
    - B. `let` variables are block-scoped, `const` variables are function-scoped.
    - C. `const` variables cannot be reassigned, `let` variables can.
    - D. `const` and `let` have no difference in TypeScript.

14. **Which of the following will cause an error?**

    ```typescript
    const a = 10;
    a = 20;
    ```

    - A. The code will run without errors.
    - B. `const` variables cannot be reassigned.
    - C. `a` is not defined.
    - D. Syntax error

Modules

15. **How do you export a function from a module in TypeScript?**

    ```typescript
    function greet() {
        console.log("Hello");
    }
    ```

    - A. `module.exports = greet;`
    - B. `export function greet() { console.log("Hello"); }`
    - C. `exports.greet = greet;`
    - D. `export greet;`

16. **How do you import a function from a module in TypeScript?**

    ```typescript
    import { greet } from './module';
    ```

    - A. `const greet = require('./module');`

- B. `import greet from './module';`
- C. `import { greet } from './module';`
- D. `import * as greet from './module';`

Native ECMAScript Modules

17. **Which statement correctly imports the default export from a module in TypeScript?**

```typescript
import moduleName from './module';
```

- A. `import { moduleName } from './module';`
- B. `import * as moduleName from './module';`
- C. `import default from './module';`
- D. `import moduleName from './module';`

18. **How do you export a default function in TypeScript?**

```typescript
function greet() {
    console.log("Hello");
}
```

- A. `export = greet;`
- B. `export default greet;`
- C. `exports.greet = greet;`
- D. `export greet;`

Import Inquirer ECMAScript Module

19. **How do you import `inquirer` in TypeScript as an ECMAScript module?**

- A. `const inquirer = require('inquirer');`
- B. `import inquirer from 'inquirer';`
- C. `import * as inquirer from 'inquirer';`
- D. `require('inquirer');`

20. **Which version of Node.js supports native ECMAScript modules?**
- A. Node.js v10
- B. Node.js v12
- C. Node.js v14
- D. Node.js v8

Chalk

21. **What is the purpose of the `chalk` library in TypeScript?**
- A. To perform HTTP requests
- B. To format dates
- C. To style terminal strings
- D. To handle JSON data

22. **How do you use `chalk` to make text red in TypeScript?**
```typescript
import chalk from 'chalk';
```
- A. `console.log(chalk.red("Hello"));`
- B. `console.log(chalk("Hello").red);`
- C. `console.log(chalk("Hello", "red"));`
- D. `console.log(chalk.color("red", "Hello"));`

Unions and Literals

23. **Which of the following is a valid union type in TypeScript?**

    ```typescript
    let value: string | number;
    ```

    - A. `let value: string | boolean;`
    - B. `let value: string | number;`
    - C. `let value: string & number;`
    - D. `let value: number & boolean;`

24. **How do you define a literal type in TypeScript?**

    ```typescript
    let direction: "left" | "right";
    ```

    - A. `let direction: string;`
    - B. `let direction: "left" | "right";`
    - C. `let direction: string | number;`
    - D. `let direction: any;`

Objects

25. **How do you define an object type in TypeScript?**

    ```typescript
    let person: { name: string; age: number };
    ```

    - A. `let person: object;`
    - B. `let person: { name: string; age: number };`
    - C. `let person: { name: any; age: any };`
    - D. `let person: any;`

26. **What will be the output of the following TypeScript code?**

```typescript
const obj = { a: 1, b: 2 };
console.log(obj.c);
```

- A. `undefined`
- B. `null`
- C. `Error: c is not defined`
- D. `0`

Object Aliased

27. **What is the purpose of type aliases in TypeScript?**

- A. To create new types from existing types
- B. To rename existing types
- C. To define union and intersection types
- D. To assign default values to types

28. **How do you create a type alias for an object in TypeScript?**

```typescript
type Person = { name: string; age: number };
```

- A. `alias Person = { name: string; age: number };`
- B. `type Person = { name: string; age: number };`
- C. `const Person = { name: string; age: number };`
- D. `let Person = { name: string; age: number };`

Structural Typing and Object Literals

29. **What is structural typing in TypeScript?**
    - A. Types are defined by their names
    - B. Types are defined by their structure
    - C. Types are checked at runtime
    - D. Types are ignored during compilation

30. **Which of the following will result in a type error?**
    ```typescript
    interface Point {
        x: number;
        y: number;
    }
    const point: Point = { x: 10, y: 20, z: 30 };
    ```
    - A. The code will run without errors
    - B. `Point` cannot have extra properties
    - C. `z` is not defined in `Point`
    - D. TypeScript allows extra properties

Nested Objects

31. **How do you define a nested object type in TypeScript?**
    ```typescript
    interface Address {
        street: string;
        city: string;
    ```

```
}
interface Person {
  name: string;
  address: Address;
}
```

- A. `interface Person { name: string; address: object; }`
- B. `interface Person { name: string; address: Address; }`
- C. `interface Person { name: string; address: { street: string; city: string; }; }`
- D. `interface Person { name: string; address: any; }`

32. **How do you access a nested object property in TypeScript?**

```typescript
const person = {
  name: "John",
  address: {
    street: "123 Main St",
    city: "Anytown"
  }
};
```

- A. `person.address.street`
- B. `person["address"]["street"]`
- C. Both A and B
- D. Neither A nor B

Intersection Types

33. **What is an intersection type in TypeScript?**

- A. A type that combines multiple types into one
- B. A type that can be one of several types
- C. A type that is defined by its structure
- D. A type that extends another type

34. **Which of the following defines an intersection type in TypeScript?**

```typescript
type Combined = TypeA & TypeB;
```

- A. `type Combined = TypeA | TypeB;`
- B. `type Combined = TypeA & TypeB;`
- C. `type Combined = { ...TypeA, ...TypeB };`
- D. `type Combined = TypeA + TypeB;`

any, unknown, and never Types

35. **What is the key difference between `any` and `unknown` in TypeScript?**
- A. `any` type can be assigned any value, `unknown` cannot.
- B. `unknown` type can be assigned any value, `any` cannot.
- C. `any` type bypasses type checking, `unknown` requires type assertions.
- D. `unknown` type bypasses type checking, `any` requires type assertions.

36. **When should you use the `never` type in TypeScript?**
- A. For variables that can hold any value
- B. For variables that should never hold a value
- C. For functions that never return
- D. For functions that always return a value

Explicit Casting

37. **How do you explicitly cast a value in TypeScript?**

```typescript
let value: any = "Hello";
```

- A. `let str: string = value as string;`
- B. `let str: string = <string>value;`
- C. Both A and B
- D. Neither A nor B

38. **What is the purpose of explicit casting in TypeScript?**

- A. To change the value of a variable
- B. To change the type of a variable
- C. To bypass type checking
- D. To convert a value to another type

Enums

39. **How do you define an enum in TypeScript?**

```typescript
enum Color {
    Red,
    Green,
    Blue
}
```

- A. `enum Color = { Red, Green, Blue }`
- B. `enum Color { Red, Green, Blue }`
- C. `const Color = { Red, Green, Blue }`

- D. `type Color = { Red, Green, Blue }`

40. **What is the default value of the first member of an enum in TypeScript?**
    - A. `0`
    - B. `1`
    - C. `undefined`
    - D. `null`

const Enums

41. **How do you define a `const enum` in TypeScript?**
    ```typescript
    const enum Direction {
        Up,
        Down,
        Left,
        Right
    }
    ```
    - A. `const enum Direction { Up, Down, Left, Right }`
    - B. `enum Direction { Up, Down, Left, Right }`
    - C. `type Direction = { Up, Down, Left, Right }`
    - D. `const Direction = { Up, Down, Left, Right }`

42. **What is the advantage of using `const enum` over regular enums in TypeScript?**
    - A. Better performance
    - B. Easier to read
    - C. Compile-time optimization
    - D. Runtime type checking

Arrays

43. **How do you define an array of numbers in TypeScript?**

    ```typescript
    let numbers: number[] = [1, 2, 3];
    ```

    - A. `let numbers: Array<number> = [1, 2, 3];`
    - B. `let numbers: number[] = [1, 2, 3];`
    - C. Both A and B
    - D. Neither A nor B

44. **What is the output of the following TypeScript code?**

    ```typescript
    let arr: number[] = [1, 2, 3];
    console.log(arr[3]);
    ```

    - A. `undefined`
    - B. `null`
    - C. `0`
    - D. `Error: Index out of bounds`

Functions

45. **How do you define a function in TypeScript that returns a number?**

    ```typescript
    function add(a: number, b: number): number {
        return a + b;
    }
    ```

```

- A. `function add(a, b): number { return a + b; }`
- B. `function add(a: number, b: number): number { return a + b; }`
- C. `function add(a: number, b: number) { return a + b; }`
- D. `function add(a, b) { return a + b; }`

46. **Which of the following is a correct way to define an arrow function in TypeScript?**

```typescript
const greet = (name: string): string => `Hello, ${name}`;
```

- A. `const greet = (name: string): string => { return `Hello, ${name}`; }`
- B. `const greet = (name): string => `Hello, ${name}`;`
- C. `const greet = (name: string) => `Hello, ${name}`;`
- D. `const greet = name: string => `Hello, ${name}`;`

Function Optional Parameter

47. **How do you define an optional parameter in a TypeScript function?**

```typescript
function greet(name?: string) {
    console.log(`Hello, ${name || 'Guest'}`);
}
```

- A. `function greet(name: string?) { ... }`
- B. `function greet(name: ?string) { ... }`
- C. `function greet(name?: string) { ... }`
- D. `function greet(?name: string) { ... }`

48. **What is the output of the following TypeScript code?**
```

```typescript
function greet(name?: string) {
    console.log(`Hello, ${name || 'Guest'}`);
}
greet();
```

- A. `Hello, `
- B. `Hello, Guest`
- C. `Hello, undefined`
- D. `Error: Missing argument`

Function Default Parameter

49. **How do you define a default parameter in a TypeScript function?**

```typescript
function greet(name: string

 = 'Guest') {
    console.log(`Hello, ${name}`);
}
```

- A. `function greet(name = 'Guest') { ... }`
- B. `function greet(name: string = 'Guest') { ... }`
- C. `function greet(name: string?) { ... }`
- D. `function greet(name?: string = 'Guest') { ... }`

50. **What is the output of the following TypeScript code?**

```typescript
function greet(name: string = 'Guest') {
```

```
    console.log(`Hello, ${name}`);
}
greet();
```

- A. `Hello, `
- B. `Hello, Guest`
- C. `Hello, undefined`
- D. `Error: Missing argument`

Function Rest Parameter

51. **How do you define a rest parameter in a TypeScript function?**

```typescript
function sum(...numbers: number[]): number {
    return numbers.reduce((acc, curr) => acc + curr, 0);
}
```

- A. `function sum(...numbers: number[]) { ... }`
- B. `function sum(numbers: number[]) { ... }`
- C. `function sum(...numbers: [number]) { ... }`
- D. `function sum(...numbers: any[]) { ... }`

52. **What is the output of the following TypeScript code?**

```typescript
function sum(...numbers: number[]): number {
    return numbers.reduce((acc, curr) => acc + curr, 0);
}
console.log(sum(1, 2, 3));
```

- A. `1`
- B. `3`
- C. `6`
- D. `undefined`

Async

53. **How do you define an async function in TypeScript?**

```typescript
async function fetchData(): Promise<string> {
    return "data";
}
```

- A. `async function fetchData(): Promise<string> { ... }`
- B. `function fetchData(): Promise<string> { ... }`
- C. `async function fetchData(): string { ... }`
- D. `function fetchData(): string { ... }`

54. **What is the output of the following TypeScript code?**

```typescript
async function fetchData(): Promise<string> {
    return "data";
}
fetchData().then(result => console.log(result));
```

- A. `data`
- B. `undefined`
- C. `Promise { "data" }`
- D. `Error: fetchData is not defined`

Function Overloads

55. **How do you define function overloads in TypeScript?**

    ```typescript
    function add(a: number, b: number): number;
    function add(a: string, b: string): string;
    function add(a: any, b: any): any {
        return a + b;
    }
    ```

    - A. `function add(a: number, b: number): number;`
    - B. `function add(a: string, b: string): string;`
    - C. `function add(a: any, b: any): any;`
    - D. All of the above

56. **What is the purpose of function overloads in TypeScript?**

    - A. To define multiple implementations of a function
    - B. To handle different types of arguments in a function
    - C. To improve code readability
    - D. To avoid type checking

Tuples

57. **How do you define a tuple in TypeScript?**

    ```typescript
    let tuple: [number, string] = [1, "one"];
    ```

    - A. `let tuple: [number, string] = [1, "one"];`

- B. `let tuple: Array<number, string> = [1, "one"];`
- C. `let tuple: [number, string] = [1, 1];`
- D. `let tuple: [any] = [1, "one"];`

58. **What is the output of the following TypeScript code?**

```typescript
let tuple: [number, string] = [1, "one"];
console.log(tuple[0]);
```

- A. `1`
- B. `one`
- C. `undefined`
- D. `null`

Additional Advanced Level Questions

59. **What is the difference between `interface` and `type` aliases in TypeScript?**
- A. `interface` can define a structure, `type` can define a type
- B. `interface` can be merged, `type` cannot
- C. `type` can be used for unions, `interface` cannot
- D. All of the above

60. **Which TypeScript utility type creates a type consisting of all properties of T set to optional?**
- A. `Partial<T>`
- B. `Readonly<T>`
- C. `Required<T>`
- D. `Record<K, T>`

61. **What is the purpose of `keyof` in TypeScript?**

- A. To get the type of the keys of an object
- B. To get the values of an object
- C. To iterate over the keys of an object
- D. To create a mapped type

62. **Which TypeScript feature allows checking if a property exists in an object at runtime?**
- A. `type guards`
- B. `type assertions`
- C. `keyof`
- D. `in operator`

63. **What does the following TypeScript code do?**

```typescript
type NonNullable<T> = T extends null | undefined ? never : T;
```

- A. Creates a type that excludes `null` and `undefined`
- B. Creates a type that includes `null` and `undefined`
- C. Creates a type that is nullable
- D. Creates a type that is non-nullable

64. **Which utility type constructs a type by picking the set of properties K from T?**
- A. `Pick<T, K>`
- B. `Omit<T, K>`
- C. `Extract<T, U>`
- D. `Exclude<T, U>`

65. **How do you handle mixed-type arrays in TypeScript?**

```typescript
let arr: (number | string)[] = [1, "one", 2, "two"];
```

```

- A. `let arr: Array<number | string> = [1, "one", 2, "two"];`
- B. `let arr: (number | string)[] = [1, "one", 2, "two"];`
- C. Both A and B
- D. Neither A nor B

66. **How do you define a readonly property in a TypeScript interface?**

```typescript
interface Person {
    readonly name: string;
    age: number;
}
```

- A. `readonly name: string`
- B. `const name: string`
- C. `name: readonly string`
- D. `name: string const`

67. **Which TypeScript type utility makes all properties in T readonly?**

- A. `Readonly<T>`
- B. `Partial<T>`
- C. `Required<T>`
- D. `Record<K, T>`

68. **How can you iterate over the keys of an object type in TypeScript?**

```typescript
type Keys = keyof { name: string; age: number; };
```

- A. `type Keys = keyof { name: string; age: number; };`
```

- B. `type Keys = typeof { name: string; age: number; };`
- C. `type Keys = Object.keys({ name: string; age: number; });`
- D. `type Keys = Object.values({ name: string; age: number; });`

69. **What will be the output of the following TypeScript code?**

```typescript
enum Direction {
  Up,
  Down,
  Left,
  Right
}
console.log(Direction.Left);
```

- A. `Left`
- B. `2`
- C. `undefined`
- D. `Error: Direction is not defined`

70. **Which utility type extracts the type of the elements of an array type T?**

- A. `ArrayElement<T>`
- B. `ElementType<T>`
- C. `TypeOfArray<T>`
- D. `Array<T>`

71. **What does the `Partial<T>` utility type do in TypeScript?**

- A. Creates a type with all properties of T optional
- B. Creates a type with all properties of T required
- C. Creates a type with all properties of T readonly

- D. Creates a type with all properties of T nullable

72. **How do you ensure a function argument can be of multiple types in TypeScript?**

```typescript
function combine(a: number | string) { ... }
```

- A. `function combine(a: number | string) { ... }`
- B. `function combine(a: number & string) { ... }`
- C. `function combine(a: any) { ... }`
-

D. `function combine(a: number) { ... } function combine(a: string) { ... }`

73. **Which of the following is a valid TypeScript tuple type?**

- A. `[number, string, boolean]`
- B. `Array<number | string | boolean>`
- C. `(number, string, boolean)`
- D. `[number | string | boolean]`

74. **What will be the output of the following TypeScript code?**

```typescript
const enum Size {
    Small,
    Medium,
    Large
}
console.log(Size.Medium);
```

- A. `Medium`

- B. `1`
- C. `undefined`
- D. `Error: Size is not defined`

75. **Which TypeScript utility type creates a type that excludes keys K from T?**
- A. `Omit<T, K>`
- B. `Pick<T, K>`
- C. `Extract<T, K>`
- D. `Exclude<T, K>`

76. **How do you define an optional property in a TypeScript interface?**
```typescript
interface Person {
  name?: string;
  age: number;
}
```
- A. `name?: string`
- B. `name: string?`
- C. `name: ?string`
- D. `?name: string`

77. **What is the output of the following TypeScript code?**
```typescript
const obj = { name: "Alice", age: 25 };
type Person = typeof obj;
```
- A. `name: string; age: number`
- B. `name: "Alice"; age: 25`

- C. `string; number`

- D. `typeof obj`

78. **How do you define a function type in TypeScript?**

```typescript
type Greet = (name: string) => string;
```

- A. `type Greet = (name: string) => string;`

- B. `type Greet = { (name: string): string };`

- C. `type Greet = (name: string): string;`

- D. `type Greet = { (name: string) => string };`

79. **What will be the output of the following TypeScript code?**

```typescript
interface Point {
    x: number;
    y: number;
}
const point: Point = { x: 10, y: 20 };
console.log(point.x);
```

- A. `10`

- B. `20`

- C. `{ x: 10, y: 20 }`

- D. `undefined`

80. **Which utility type creates a type with all properties of T required?**

- A. `Required<T>`

- B. `Readonly<T>`

- C. `Partial<T>`
- D. `Record<K, T>`

81. **How do you define a mapped type in TypeScript?**

```typescript
type Readonly<T> = { readonly [P in keyof T]: T[P] };
```

- A. `type Readonly<T> = { readonly [P in keyof T]: T[P] };`
- B. `type Readonly<T> = { readonly [P: keyof T]: T[P] };`
- C. `type Readonly<T> = { readonly [P in keyof T]: T };`
- D. `type Readonly<T> = { readonly [P: keyof T]: T };`

82. **Which utility type constructs a type by excluding null and undefined from T?**

- A. `NonNullable<T>`
- B. `Nullable<T>`
- C. `Nullish<T>`
- D. `NotNullable<T>`

83. **What will be the output of the following TypeScript code?**

```typescript
type Person = { name: string; age?: number };
const person: Person = { name: "Alice" };
console.log(person.age);
```

- A. `undefined`
- B. `null`
- C. `0`
- D. `Error: age is missing`

84. **Which TypeScript utility type constructs a type by picking all properties from T that are assignable to U?**

- A. `Extract<T, U>`
- B. `Exclude<T, U>`
- C. `Pick<T, U>`
- D. `Omit<T, U>`

85. **What is the purpose of the `as const` assertion in TypeScript?**

- A. To create a readonly array or object
- B. To create a constant value
- C. To create a mutable array or object
- D. To create a variable with a constant type

86. **Which TypeScript utility type creates a type consisting of all properties of T set to required?**

- A. `Required<T>`
- B. `Readonly<T>`
- C. `Partial<T>`
- D. `Record<K, T>`

87. **What will be the output of the following TypeScript code?**

```typescript
type Person = { name: string; age?: number };
const person: Person = { name: "Alice" };
console.log(person.age);
```

- A. `undefined`
- B. `null`
- C. `0`
- D. `Error: age is missing`

88. **Which utility type creates a type that includes only the keys of T that are assignable to U?**

- A. `Extract<T, U>`
- B. `Exclude<T, U>`
- C. `Pick<T, U>`
- D. `Omit<T, U>`

89. **How do you handle optional properties in a TypeScript interface?**

```typescript
interface Person {
    name: string;
    age?: number;
}
```

- A. `name: string; age?: number`
- B. `name: string; age: number?`
- C. `name: string?; age: number`
- D. `name?: string; age: number`

90. **What is the purpose of the `keyof` operator in TypeScript?**

- A. To get the type of the keys of an object
- B. To get the type of the values of an object
- C. To get the length of an object
- D. To get the type of an object

91. **Which TypeScript utility type constructs a type by excluding from T all properties that are assignable to U?**

- A. `Exclude<T, U>`
- B. `Extract<T, U>`

- C. `Pick<T, U>`
- D. `Omit<T, U>`

92. **What is the output of the following TypeScript code?**

```typescript
type Person = { name: string; age?: number };
const person: Person = { name: "Alice" };
console.log(person.age);
```

- A. `undefined`
- B. `null`
- C. `0`
- D. `Error: age is missing`

93. **How do you define a readonly property in a TypeScript interface?**

```typescript
interface Person {
    readonly name: string;
    age: number;
}
```

- A. `readonly name: string`
- B. `const name: string`
- C. `name: readonly string`
- D. `name: string const`

94. **What is the output of the following TypeScript code?**

```typescript
interface Person {
```

```
    name: string;
    age?: number;
}
const person: Person = { name: "Alice" };
console.log(person.age);
```

- A. `undefined`
- B. `null`
- C. `0`
- D. `Error: age is missing`

95. **Which TypeScript utility type creates a type consisting of all properties of T set to optional?**
    - A. `Partial<T>`
    - B. `Readonly<T>`
    - C. `Required<T>`
    - D. `Record<K, T>`

96. **How do you define a readonly array in TypeScript?**
    ```typescript
    let arr: ReadonlyArray<number> = [1, 2, 3];
    ```
    - A. `let arr: ReadonlyArray<number> = [1, 2, 3];`
    - B. `let arr: Array<Readonly<number>> = [1, 2, 3];`
    - C. `let arr: Array<number> = [1, 2, 3];`
    - D. `let arr: readonly number[] = [1, 2, 3];`

97. **What is the output of the following TypeScript code?**
    ```typescript
    interface Person {
    ```

```
    name: string;
    age?: number;
}
const person: Person = { name: "Alice" };
console.log(person.age);
```

- A. `undefined`
- B. `null`
- C. `0`
- D. `Error: age is missing`

98. **Which utility type creates a type that includes only the keys of T that are assignable to U?**
    -

A. `Extract<T, U>`
- B. `Exclude<T, U>`
- C. `Pick<T, U>`
- D. `Omit<T, U>`

99. **How do you handle optional properties in a TypeScript interface?**

```typescript
interface Person {
    name: string;
    age?: number;
}
```

- A. `name: string; age?: number`
- B. `name: string; age: number?`
- C. `name: string?; age: number`

- D. `name?: string; age: number`

100. **What is the purpose of the `keyof` operator in TypeScript?**

- A. To get the type of the keys of an object
- B. To get the type of the values of an object
- C. To get the length of an object
- D. To get the type of an object