Important Questions / Answers

Certainly! Here are 100 important TypeScript questions along with their corresponding coding solutions:

Basic Concepts

1. **What is TypeScript?**
   - TypeScript is a statically typed superset of JavaScript that compiles to plain JavaScript.
2. **How do you install TypeScript globally?**

```
npm install -g typescript
```

3. **How do you compile a TypeScript file?**

```
tsc filename.ts
```

4. **What is a TypeScript configuration file used for?**
   - The `tsconfig.json` file specifies the root files and compiler options for the TypeScript project.
5. **How do you initialize a TypeScript project?**

```
tsc --init
```

Types

6. **What are the basic types in TypeScript?**
   - `boolean`, `number`, `string`, `array`, `tuple`, `enum`, `any`, `void`, `null`, `undefined`, `never`.

7. **How do you define a boolean variable?**

```
let isDone: boolean = false;
```

8. **What is type inference in TypeScript?**
   - TypeScript can infer types based on the value assigned to a variable.

9. **What is an interface?**
   - An interface defines the structure of an object in TypeScript.

10. **How do you implement an interface in a class?**

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

class Student implements Person {
  name: string;
  age: number;
  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }
}
```

Functions

11. **How do you define a function with typed parameters and return type?**

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

12. **What are optional parameters in TypeScript?**

```
function greet(name: string, greeting?: string): string {
  return greeting ? `${greeting}, ${name}` : `Hello, ${name}`;
}
```

13. **What are default parameters in TypeScript?**

```
function greet(name: string, greeting: string = "Hello"): string {
  return `${greeting}, ${name}`;
}
```

14. **How do you define a function type?**

```
type MathFunction = (a: number, b: number) => number;
let add: MathFunction = (a, b) => a + b;
```

Arrays and Tuples

15. **How do you define a typed array?**

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

16. **What is a tuple in TypeScript?**

```
let person: [string, number] = ["Alice", 30];
```

17. **How do you define an array of objects?**

```
interface Person {
  name: string;
  age: number;
}
let people: Person[] = [{ name: "Bob", age: 25 }, { name: "Alice", age:
30 }];
```

18. **How do you create a readonly array in TypeScript?**

```
let readonlyNumbers: ReadonlyArray<number> = [1, 2, 3];
```

Enums

19. **How do you define a basic enum?**

```
enum Direction {
  Up,
  Down,
  Left,
  Right
}
```

20. **How do you assign values to enum members?**

```
enum Direction {
  Up = 1,
  Down,
  Left,
  Right
}
```

21. **What is a string enum?**

```
enum Direction {
  Up = "UP",
  Down = "DOWN",
  Left = "LEFT",
  Right = "RIGHT"
}
```

Type Aliases

22. **How do you create a type alias in TypeScript?**

```
type StringOrNumber = string | number;
```

23. **How do you define a complex type alias?**

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

24. **How do you use a type alias in a function?**

```
function printId(id: StringOrNumber): void {
  console.log(id);
}
```

Union and Intersection Types

25. **What is a union type?**

```
let id: string | number;
```

26. **What is an intersection type?**

```
interface A {
  a: number;
}
interface B {
  b: number;
}
type AB = A & B;
```

27. **How do you check for types at runtime?**

```
function logType(arg: any) {
  if (typeof arg === "number") {
    console.log("It's a number");
  } else if (typeof arg === "string") {
    console.log("It's a string");
  }
}
```

Classes and Interfaces

28. **How do you define a class in TypeScript?**

```
class Animal {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
  move(distance: number = 0) {
    console.log(`${this.name} moved ${distance}m.`);
  }
}
```

29. **How do you extend a class in TypeScript?**

```
class Dog extends Animal {
  bark() {
    console.log("Woof! Woof!");
  }
}
```

30. **How do you implement an interface in a class?**

```
interface Shape {
  area(): number;
}

class Circle implements Shape {
  radius: number;
  constructor(radius: number) {
    this.radius = radius;
  }
  area(): number {
    return Math.PI * this.radius ** 2;
  }
}
```

Generics

31. **What are generics in TypeScript?**
    - Generics allow you to create reusable components that can work with a variety of types.
32. **How do you define a generic function?**

```
function identity<T>(arg: T): T {
  return arg;
}
```

33. **How do you use generics with interfaces?**

```
interface GenericIdentityFn<T> {
  (arg: T): T;
}
let identity: GenericIdentityFn<number> = (arg: number) => arg;
```

34. **How do you specify constraints for generics?**

```
interface Lengthwise {
  length: number;
}

function loggingIdentity<T extends Lengthwise>(arg: T): T {
  console.log(arg.length);
  return arg;
}
```

Decorators

35. **What are decorators in TypeScript?**
    - Decorators are a design pattern used to add metadata or modify classes and class members.
36. **How do you define a class decorator?**

```
function sealed(constructor: Function) {
  Object.seal(constructor);
  Object.seal(constructor.prototype);
}

@sealed
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
  greet() {
    return "Hello, " + this.greeting;
  }
}
```

37. **How do you define a method decorator?**

```
function enumerable(value: boolean) {
  return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {
    descriptor.enumerable = value;
  };
}

class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }

  @enumerable(false)
  greet() {
    return "Hello, " + this.greeting;
  }
}
```

Modules

38. **How do you export a module in TypeScript?**

```
// utils.ts
export function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import { add } from "./utils";
let result = add(1, 2);
```

39. **How do you import a module in TypeScript?**

```
import { add } from "./utils";
let result = add(1, 2);
```

40. **How do you export default in TypeScript?**

```
// utils.ts
export default function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import add from "./utils";
```

```
let result = add(1, 2);
```

Promises and Async/Await

41. **How do you create a Promise in TypeScript?**

```
function fetchData(): Promise<string> {
  return new Promise((resolve, reject) => {
    // Fetch data
    let data = "Some data fetched";
    resolve(data);
  });
}
```

42. **How do you use async/await with a Promise?**

```
async function fetchDataAsync() {
  let data = await fetchData();
  console.log(data);
}
```

43. **How do you handle errors with async/await?**

```
async function fetchDataAsync() {
  try {
    let data = await fetchData();
    console.log(data);
  } catch (error) {
    console.error("Error fetching data:", error);
  }
}
```

Type Guards and Type Assertions

44. **What is a type guard in TypeScript?**

```
function isNumber(x: any): x is number {
  return typeof x === "number";
}
```

45. **How do you use a type guard?**

```
function example(x: any) {
  if (isNumber(x)) {
    console.log(x.toFixed(2));
  } else {
    console.log(x.toUpperCase());
  }
}
```

46. **What is a type assertion in TypeScript?**

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

Intersection Types

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

```
interface A {
  a: number;
}
interface B {
  b: number;
}
type AB = A & B;
```

Conditional Types

48. **What are conditional types in TypeScript?**

```
type IsString<T> = T extends string ? "yes" : "no";
let result: IsString<string> = "yes";
```

Utility Types

49. **What are utility types in TypeScript?**

```
interface Person {
  name: string;
  age: number;
}
type ReadonlyPerson = Readonly<Person>;
```

50. **How do you use the `Partial` utility type?**

```
interface Todo {
  title: string;
  description: string;
}
function updateTodo(todo: Todo, fieldsToUpdate: Partial<Todo>) {
  return { ...todo, ...fieldsToUpdate };
}
```

keyof Operator

51. **What is the `keyof` operator in TypeScript?**

```
interface Person {
  name: string;
  age: number;
}
type PersonKey = keyof Person;
```

Mapped Types

52. **What are mapped types in TypeScript?**

```
interface Person {
  name: string;
  age: number;
}
type ReadonlyPerson<T> = {
  readonly [P in keyof T]: T[P];
};
```

Type Guards

53. **How do you define a type guard function?**

```
function isNumber(x: any): x is number {
  return typeof x === "number";
}
```

Declaration Merging

54. **What is declaration merging in TypeScript?**

```
interface Box {
  height: number;
  width: number;
}
interface Box {
  scale: number;
}
```

Namespaces

55. **What are namespaces in TypeScript?**

```
namespace Geometry {
  export interface Vector2D {
    x: number;
    y: number;
  }
}
```

JSX and React

56. **How do you use JSX in TypeScript?**

```
interface Props {
  name: string;
}
```

```
const App = ({ name }: Props) => <div>Hello, {name}!</div>;
```

Ambient Declarations

57. **What are ambient declarations in TypeScript?**

```
declare var jQuery: (selector: string) => any;
```

Type Checking

58. **How does TypeScript perform type checking?**
    - TypeScript performs static type checking during compilation to detect type errors.

Inference

59. **How does TypeScript infer types?**
    - TypeScript uses type inference to determine the types of variables based on their usage.

Casting

60. **How do you cast types in TypeScript?**

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

Modules

61. **How do you export a module in TypeScript?**

```
// utils.ts
export function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import { add } from "./utils";
let result = add(1, 2);
```

62. **How do you import a module in TypeScript?**

```
import { add } from "./utils";
let result = add(1, 2);
```

Classes

63. **How do you define a class in TypeScript?**

```
class Animal {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
  move(distance: number = 0) {
    console.log(`${this.name} moved ${distance}m.`);
  }
}
```

Inheritance

64. **How do you extend a class in TypeScript?**

```
class Dog extends Animal {
  bark() {
    console.log("Woof! Woof!");
  }
}
```

Abstract Classes

65. **What are abstract classes in TypeScript?**

```
abstract class Animal {
  abstract makeSound(): void;
  move(): void {
    console.log("roaming the earth...");
  }
}
```

Interfaces

66. **How do you define an interface in TypeScript?**

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

67. **How do you implement an interface in a class?**

```
class Student implements Person {
  name: string;
  age: number;
  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }
}
```

Functions

68. **How do you define a function with typed parameters and return type?**

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

69. **What are optional parameters in TypeScript?**

```
function greet(name: string, greeting?: string): string {
  return greeting ? `${greeting}, ${name}` : `Hello, ${name}`;
}
```

70. **What are default parameters in TypeScript?**

```
function greet(name: string, greeting: string = "Hello"): string {
  return `${greeting}, ${name}`;
}
```

Generics

71. **What are generics in TypeScript?**

```
function identity<T>(arg: T): T {
  return arg;
}
```

72. **How do you define a generic interface?**

```
interface GenericIdentityFn<T> {
  (arg: T): T;
}
```

Decorators

73. **What are decorators in TypeScript?**
    - Decorators are a design pattern used to add metadata or modify classes and class members.
74. **How do you define a class decorator?**

```
function sealed(constructor: Function) {
  Object.seal(constructor);
  Object.seal(constructor.prototype);
}

@sealed
```

```
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
  greet() {
    return "Hello, " + this.greeting;
  }
}
```

75. **How do you define a method decorator?**

```
function enumerable(value: boolean) {
  return function (target: any, propertyKey: string, descriptor:
PropertyDescriptor) {
    descriptor.enumerable = value;
  };
}

class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }

  @enumerable(false)
  greet() {
    return "Hello, " + this.greeting;
  }
}
```

Modules

76. **How do you export a module in TypeScript?**

```
// utils.ts
export function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import { add } from "./utils";
let result = add(1, 2);
```

77. **How do you import a module in TypeScript?**

```
import { add } from "./utils";
let result = add(1, 2);
```

78. **How do you export default in TypeScript?**

```
// utils.ts
export default function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import add from "./utils";
let result = add(1, 2);
```

Promises and Async/Await

79. **How do you create a Promise in TypeScript?**

```
function fetchData(): Promise<string> {
  return new Promise((resolve, reject) => {
    // Fetch data
    let data = "Some data fetched";
    resolve(data);
  });
}
```

80. **How do you use async/await with a Promise?**

```
async function fetchDataAsync() {
  let data = await fetchData();
  console.log(data);
}
```

81. **How do you handle errors with async/await?**

```
async function fetchDataAsync() {
  try {
    let data = await fetchData();
    console.log(data);
  } catch (error) {
    console.error("Error fetching data:", error);
  }
}
```

Type Guards and Type Assertions

82. **What is a type guard in TypeScript?**

```
function isNumber(x: any): x is number {
  return typeof x === "number";
}
```

83. **How do you use a type guard?**

```
function example(x: any) {
  if (isNumber(x)) {
    console.log(x.toFixed(2));
  } else {
    console.log(x.toUpperCase());
  }
}
```

84. **What is a type assertion in TypeScript?**

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

Intersection Types

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

```
interface A {
  a: number;
}
interface B {
  b: number;
}
type AB = A & B;
```

Conditional Types

86. **What are conditional types in TypeScript?**

```
type IsString<T> = T extends string ? "yes" : "no";
let result: IsString<string> = "yes";
```

Utility Types

87. **What are utility types in TypeScript?**

```
interface Person {
  name: string;
  age: number;
}
type ReadonlyPerson = Readonly<Person>;
```

88. **How do you use the `Partial` utility type?**

```
interface Todo {
  title: string;
  description: string;
}
function updateTodo(todo: Todo, fieldsToUpdate: Partial<Todo>) {
  return { ...todo, ...fieldsToUpdate };
}
```

keyof Operator

89. **What is the `keyof` operator in TypeScript?**

```
interface Person {
  name: string;
  age: number;
}
type PersonKey = keyof Person;
```

Mapped Types

90. **What are mapped types in TypeScript?**

```
interface Person {
  name: string;
  age: number;
}
type ReadonlyPerson<T> = {
  readonly [P in keyof T]: T[P];
};
```

Type Guards

91. **How do you define a type guard function?**

```
function isNumber(x: any): x is number {
  return typeof x === "number";
}
```

Declaration Merging

92. **What is declaration merging in TypeScript?**

```
interface Box {
  height: number;
  width: number;
}
interface Box {
  scale: number;
}
```

Namespaces

93. **What are namespaces in TypeScript?**

```
namespace Geometry {
  export interface Vector2D {
    x: number;
    y: number;
  }
}
```

JSX and React

94. **How do you use JSX in TypeScript?**

```
interface Props {
  name: string;
}
const App = ({ name }: Props) => <div>Hello, {name}!</div>;
```

Ambient Declarations

95. **What are ambient declarations in TypeScript?**

```
declare var jQuery: (selector: string) => any;
```

Type Checking

96. **How does TypeScript perform type checking?**
    - TypeScript performs static type checking during compilation to detect type errors.

Inference

97. **How does TypeScript infer types?**
    - TypeScript uses type inference to determine the types of variables based on their usage.

Casting

98. **How do you cast types in TypeScript?**

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

Modules

99. **How do you export a module in TypeScript?**

```
// utils.ts
export function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import { add } from "./utils";
let result = add(1, 2);
```

100. **How do you import a module in TypeScript?**

```
import { add } from "./utils";
let result = add(1, 2);
```