1. What is TypeScript?

- A) A framework for building web applications

- B) A superset of JavaScript that adds static typing

- C) A library for creating user interfaces

- D) A database query language

- Answer: B) A superset of JavaScript that adds static typing

Explanation: TypeScript extends JavaScript by adding static type definitions, which help developers identify potential errors during development rather than at runtime.

2. Which command is used to compile a TypeScript file?

- A) `tsc`

- B) `ts`

- C) `compile-ts`

- D) `transpile-ts`

- Answer: A) `tsc`

Explanation: The `tsc` (TypeScript Compiler) command is used to compile TypeScript files into JavaScript files.

3. How do you declare a variable with a specific type in TypeScript?

- A) `let variable = "value"`

- B) `let variable: type = value`

- C) `let variable: value`

- D) `let variable: type;`

- Answer: B) `let variable: type = value`

Explanation: In TypeScript, you declare a variable with a specific type using the syntax `let variable: type = value`.

4. What is the default type of a variable in TypeScript if the type is not specified?

- A) `any`

- B) `string`

- C) `number`

- D) `undefined`

- Answer: A) `any`

Explanation: If the type is not specified, TypeScript assigns the `any` type to the variable, allowing it to hold values of any type.

5. Which of the following is not a TypeScript data type?

- A) `string`

- B) `boolean`

- C) `tuple`

- D) `object`

- Answer: D) `object`

Explanation: In TypeScript, the basic types are `string`, `number`, `boolean`, `array`, `tuple`, `enum`, `any`, `void`, `null`, and `undefined`. While `object` is used in JavaScript, TypeScript uses more specific types like `Record<string, any>`.

6. How do you specify an optional parameter in a function in TypeScript?

- A) `parameter?: type`

- B) `parameter: type?`

- C) `parameter: type = undefined`

- D) `parameter?: type = default`

- Answer: A) `parameter?: type`

Explanation: In TypeScript, you can specify an optional parameter using the syntax `parameter?: type`.

7. Which keyword is used to define an interface in TypeScript?

- A) `interface`

- B) `type`

- C) `class`

- D) `struct`

- Answer: A) `interface`

Explanation: The `interface` keyword is used to define an interface in TypeScript, which is a way to describe the shape of an object.

8. How do you implement an interface in a TypeScript class?

- A) `class MyClass implements MyInterface`

- B) `class MyClass extends MyInterface`

- C) `class MyClass uses MyInterface`

- D) `class MyClass inherits MyInterface`

- Answer: A) `class MyClass implements MyInterface`

Explanation: In TypeScript, a class implements an interface using the `implements` keyword.

9. What is the correct way to define a tuple in TypeScript?

- A) `let tuple: [string, number]`

- B) `let tuple: (string, number)`

- C) `let tuple: {string, number}`

- D) `let tuple: [string; number]`

- Answer: A) `let tuple: [string, number]`

Explanation: In TypeScript, a tuple is defined using square brackets with the types of each element specified, like `let tuple: [string, number]`.

10. What is the use of the `readonly` modifier in TypeScript?

- A) To create a read-only variable

- B) To prevent a variable from being reassigned

- C) To create a constant variable

- D) To create a read-only property in an object or class

- Answer: D) To create a read-only property in an object or class

Explanation: The `readonly` modifier in TypeScript is used to make a property read-only, meaning it cannot be reassigned after the initial assignment.

### Object-Oriented Programming (OOP) in TypeScript

11. How do you define a class in TypeScript?

- A) `class ClassName {}`

- B) `function ClassName() {}`

- C) `type ClassName = {}`

- D) `interface ClassName {}`

- Answer: A) `class ClassName {}`

Explanation: A class in TypeScript is defined using the `class` keyword followed by the class name and curly braces to enclose its members.

12. What is the purpose of the `constructor` method in a TypeScript class?

- A) To define a method

- B) To initialize the class's properties

- C) To define a class

- D) To implement an interface

- Answer: B) To initialize the class's properties

Explanation: The `constructor` method is used to initialize the properties of a class when an instance of the class is created.

13. How do you create an instance of a class in TypeScript?

- A) `let obj = new ClassName()`

- B) `let obj = create ClassName()`

- C) `let obj = ClassName()`

- D) `let obj = new ClassName`

- Answer: A) `let obj = new ClassName()`

Explanation: An instance of a class is created using the `new` keyword followed by the class name and parentheses.

14. How do you extend a class in TypeScript?

- A) `class Child extends Parent`

- B) `class Child inherits Parent`

- C) `class Child implements Parent`

- D) `class Child extends from Parent`

- Answer: A) `class Child extends Parent`

Explanation: A class is extended using the `extends` keyword followed by the name of the parent class.

15. How do you implement an interface in a TypeScript class?

- A) `class MyClass implements MyInterface`

- B) `class MyClass extends MyInterface`

- C) `class MyClass uses MyInterface`

- D) `class MyClass inherits MyInterface`

- Answer: A) `class MyClass implements MyInterface`

Explanation: To implement an interface, use the `implements` keyword followed by the interface name.

16. Which access modifier makes a class member accessible only within the class itself?

- A) `public`

- B) `protected`

- C) `private`

- D) `readonly`

- Answer: C) `private`

Explanation: The `private` access modifier restricts access to the class member only within the class itself.

17. How do you define an abstract class in TypeScript?

- A) `abstract class ClassName {}`

- B) `class ClassName abstract {}`

- C) `class ClassName { abstract }`

- D) `abstract ClassName class {}`

- Answer: A) `abstract class ClassName {}`

Explanation: An abstract class is defined using the `abstract` keyword followed by the class name and curly braces.

18. How do you define an abstract method in a TypeScript class?

- A) `abstract methodName(): returnType;`

- B) `abstract methodName: returnType;`

- C) `methodName(): abstract returnType;`

- D) `methodName(): returnType abstract;`

- Answer: A) `abstract methodName(): returnType;`

Explanation: An abstract method in a TypeScript class is defined using the `abstract` keyword followed by the method name and its return type.

19. Which keyword is used to access a superclass's properties and methods in TypeScript?

- A) `base`

- B) `super`

- C) `parent`

- D) `ancestor`

- Answer: B) `super`

Explanation: The `super` keyword is used to access properties and methods of the superclass in TypeScript.

20. How do you define a generic class in TypeScript?

- A) `class ClassName<T> {}`

- B) `class ClassName<type> {}`

- C) `class ClassName.generic {}`

- D) `class ClassName<type T> {}`

- Answer: A) `class ClassName<T> {}`

Explanation: A generic class in TypeScript is defined using the syntax `class Class

Name<T> {}`, where `T` is a placeholder for the type that will be specified later.

### Arrays in TypeScript

21. How do you declare an array of numbers in TypeScript?

- A) `let numbers: number[]`

- B) `let numbers: array`

- C) `let numbers: [number]`

- D) `let numbers: [number[]]`

- Answer: A) `let numbers: number[]`

Explanation: An array of numbers in TypeScript is declared using `let numbers: number[]`.

22. Which method adds an element to the end of an array?

- A) `push()`

- B) `pop()`

- C) `shift()`

- D) `unshift()`

- Answer: A) `push()`

Explanation: The `push()` method adds an element to the end of an array.

23. How do you remove the first element from an array?

- A) `pop()`

- B) `shift()`

- C) `unshift()`

- D) `splice()`

- Answer: B) `shift()`

Explanation: The `shift()` method removes the first element from an array.

24. What does the `map()` method do in an array?

- A) It changes the array elements based on a function and returns a new array.

- B) It filters the array elements based on a condition.

- C) It finds the first element that matches a condition.

- D) It sorts the array elements in ascending order.

- Answer: A) It changes the array elements based on a function and returns a new array.

Explanation: The `map()` method applies a function to each element of the array and returns a new array with the results.

25. How do you check if an array includes a certain value?

- A) `array.contains(value)`

- B) `array.has(value)`

- C) `array.includes(value)`

- D) `array.indexOf(value)`

- Answer: C) `array.includes(value)`

Explanation: The `includes()` method checks if an array includes a certain value and returns `true` or `false`.

26. Which method would you use to find the index of a specific element in an array?

- A) `findIndex()`

- B) `indexOf()`

- C) `find()`

- D) `filter()`

- Answer: B) `indexOf()`

Explanation: The `indexOf()` method returns the index of the first occurrence of a specified element in the array or -1 if it is not found.

27. How do you combine two arrays in TypeScript?

- A) `concat()`

- B) `join()`

- C) `merge()`

- D) `union()`

- Answer: A) `concat()`

Explanation: The `concat()` method is used to combine two or more arrays and returns a new array.

28. What does the `reduce()` method do?

- A) It applies a function against an accumulator and each element to reduce the array to a single value.

- B) It reduces the size of the array.

- C) It filters the array based on a condition.

- D) It maps the elements to new values.

- Answer: A) It applies a function against an accumulator and each element to reduce the array to a single value.

Explanation: The `reduce()` method executes a reducer function on each element of the array, resulting in a single output value.

29. How do you create a new array from an existing array using a condition?

- A) `filter()`

- B) `map()`

- C) `reduce()`

- D) `find()`

- Answer: A) `filter()`

Explanation: The `filter()` method creates a new array with all elements that pass a test implemented by a provided function.

30. Which of the following methods can modify the contents of an array?

- A) `map()`

- B) `forEach()`

- C) `reduce()`

- D) `splice()`

- Answer: D) `splice()`

Explanation: The `splice()` method can add or remove elements from an array and thus modifies the array's contents.

### Advanced TypeScript Concepts

31. How do you declare a variable in TypeScript with a specific type and a value?

- A) `let variable = value: type`

- B) `let variable: type = value`

- C) `let variable: type; variable = value`

- D) `let variable = value; variable: type`

- Answer: B) `let variable: type = value`

Explanation: You declare a variable with a specific type and value using the syntax `let variable: type = value`.

32. Which keyword is used to export a member from a module in TypeScript?

- A) `export`

- B) `exported`

- C) `exports`

- D) `module.exports`

- Answer: A) `export`

Explanation: The `export` keyword is used to export a member (variable, function, class, etc.) from a module in TypeScript.

33. How do you import a specific member from a module in TypeScript?

- A) `import member from 'module'`

- B) `import { member } from 'module'`

- C) `import 'module'.member`

- D) `import member of 'module'`

- Answer: B) `import { member } from 'module'`

Explanation: To import a specific member from a module, use the syntax `import { member } from 'module'`.

34. What does the `as` keyword do in TypeScript?

- A) It renames a module during import

- B) It specifies a type assertion

- C) It defines an alias for a type

- D) It specifies a default export

- Answer: B) It specifies a type assertion

Explanation: The `as` keyword is used for type assertions, which tell the TypeScript compiler to treat an entity as a different type.

35. Which of the following is a valid way to define a function in TypeScript?

- A) `function myFunction(param1: type, param2: type): returnType {}`

- B) `function myFunction(param1, param2): returnType {}`

- C) `function myFunction(param1: type, param2: type): type {}`

- D) `function myFunction(param1, param2: type): returnType {}`

- Answer: A) `function myFunction(param1: type, param2: type): returnType {}`

Explanation: The correct syntax to define a function in TypeScript includes specifying the parameter types and the return type.

36. What is the purpose of type guards in TypeScript?

- A) To enforce stricter type checks during runtime

- B) To allow type-safe access to properties of objects

- C) To narrow down the type within a conditional block

- D) To prevent type coercion

- Answer: C) To narrow down the type within a conditional block

Explanation: Type guards are used to narrow down the type of a variable within a conditional block, allowing for more precise type checks and operations.

37. How do you define a type alias in TypeScript?

- A) `type alias = definition;`

- B) `alias type = definition;`

- C) `define type alias = definition;`

- D) `alias = type definition;`

- Answer: A) `type alias = definition;`

Explanation: A type alias in TypeScript is defined using the `type` keyword followed by the alias name and the definition.

38. Which of the following is true about the `enum` type in TypeScript?

- A) It allows defining a set of named constants

- B) It enforces strict typing on string values

- C) It enables creating dynamic arrays

- D) It provides a way to define polymorphic behavior

- Answer: A) It allows defining a set of named constants

Explanation: The `enum` type in TypeScript is used to define a set of named constants, making the code more readable and maintainable.

39. How do you specify that a property can be of multiple types in TypeScript?

- A) `property: type1 or type2`

- B) `property: type1 & type2`

- C) `property: type1 | type2`

- D) `property: either(type1, type2)`

- Answer: C) `property: type1 | type2`

Explanation: To specify that a property can be of multiple types, use the union type syntax `property: type1 | type2`.

40. What is the purpose of the `abstract` keyword in TypeScript?

- A) To define a base class that cannot be instantiated directly

- B) To create a type alias

- C) To define a method that cannot be overridden

- D) To specify a static member in a class

- Answer: A) To define a base class that cannot be instantiated directly

Explanation: The `abstract` keyword is used to define a base class that cannot be instantiated directly and must be extended by other classes.

### TypeScript Functions

41. How do you define a function with an optional parameter in TypeScript?

- A) `function myFunction(param1?: type): returnType {}`

- B) `function myFunction(param1: type = undefined): returnType {}`

- C) `function myFunction(param1: type?): returnType {}`

- D) `function myFunction(optional param1: type): returnType {}`

- Answer: A) `function myFunction(param1?: type): returnType {}`

Explanation: An optional parameter in a TypeScript function is defined using the `?` symbol after the parameter name.

42. How do you define a function with default parameters in TypeScript?

- A) `function myFunction(param1: type = value): returnType {}`

- B) `function myFunction(param1 = value: type): returnType {}`

- C) `function myFunction(default param1: type = value): returnType {}`

- D) `function myFunction(param1: type) = value: returnType {}`

- Answer: A) `function myFunction(param1: type = value): returnType {}`

Explanation: A function with default parameters in TypeScript is defined using the syntax `param1: type = value`.

43. How do you specify the return type of a function in TypeScript?

- A) `function myFunction(param1: type): returnType {}`

- B) `function myFunction(param1: type) returns returnType {}`

- C) `function myFunction(param1: type): type returns {}`

- D) `function myFunction(param1: type) -> returnType {}`

- Answer: A) `function myFunction(param1: type): returnType {}`

Explanation: The return type of a function is specified after the parameter list using a colon and the return type, like `function myFunction(param1: type): returnType {}`.

44. What is an arrow function in TypeScript?

- A) A function defined using the `function` keyword

- B) A concise way to define anonymous functions using the `=>` syntax

- C) A function that returns a promise

- D) A function with no parameters

- Answer: B) A concise way to define anonymous functions using the `=>` syntax

Explanation: An arrow function is a concise way to define anonymous functions using the `=>` syntax, also known as a lambda function.

45. How do you define an arrow function in TypeScript?

- A) `let myFunction = (param1: type): returnType => { /\* function body \*/ }`

- B) `let myFunction = (param1: type) => { /\* function body \*/ }`

- C) `let myFunction = (param1: type): returnType => function { /\* function body \*/ }`

- D) `let myFunction = (param1: type) -> returnType { /\* function body \*/ }`

- Answer: B) `let myFunction = (param1: type) => { /\* function body \*/ }`

Explanation: An arrow function is defined using the `=>` syntax, like `let myFunction = (param1: type) => { /\* function body \*/ }`.

46. How do you handle the `this` context in an arrow function in TypeScript?

- A) `this` refers to the enclosing lexical context

- B) `this` refers to the global context

- C) `this` refers to the function's context

- D) `this` is not available in arrow functions

- Answer: A) `this` refers to the enclosing lexical context

Explanation: In an arrow function, `this` refers to the enclosing lexical context, unlike in regular functions where `this` can change based on how the function is called.

47. What is the purpose of the `never` type in TypeScript?

- A) To indicate a function never returns

- B) To indicate a function always returns a value

- C) To define an optional parameter

- D) To create a variable that can hold any type

- Answer: A) To indicate a function never returns

Explanation: The `never` type is used to indicate that a function never returns, usually because it always throws an error or has an infinite loop.

48. Which TypeScript type would you use for a variable that can hold either a string or a number?

- A) `string | number`

- B) `string & number`

- C) `string or number`

- D) `string, number`

- Answer: A) `string | number`

Explanation: To specify that a variable can hold either a string or a number, use the union type `string | number`.

49. How do you specify a function that returns a promise in TypeScript?

- A) `function myFunction(): Promise<returnType> {}`

- B) `function myFunction(): returnType Promise {}`

- C) `function myFunction() -> Promise<returnType> {}`

- D) `function myFunction(): Promise returnType {}`

- Answer: A) `function myFunction(): Promise<returnType> {}`

Explanation: To specify a function that returns a promise, use the syntax `function myFunction(): Promise<returnType> {}`.

50. How do you define an interface with an index signature in TypeScript?

- A) `interface MyInterface { [index: type]: valueType; }`

- B) `interface MyInterface { index(type): valueType; }`

- C) `interface MyInterface { type[index]: valueType; }`

- D) `interface MyInterface { type(index): valueType; }`

- Answer: A) `interface MyInterface { [index: type]: valueType; }`

Explanation: An index signature in an interface is defined using the syntax `[index: type]: valueType`.

### TypeScript Classes and Interfaces

51. How do you declare a property in a TypeScript class?

- A) `property: type`

- B) `var property: type`

- C) `let property: type`

- D) `property(type)`

- Answer: A) `property: type`

Explanation: A property in a TypeScript class is declared using the syntax `property: type`.

52. How do you declare a method in a TypeScript class?

- A) `method(): returnType {}`

- B) `function method(): returnType {}`

- C) `method: function(): returnType {}`

- D) `function(): returnType method {}`

- Answer: A) `method(): returnType {}`

Explanation: A method in a TypeScript class is declared using the syntax `method(): returnType {}`.

53. What is the purpose of the `static` keyword in TypeScript classes?

- A) To define a method that cannot be overridden

- B) To define a property or method that belongs to the class, not instances

- C) To define an abstract method

- D) To define a read-only property

- Answer: B) To define a property or method that belongs to the class, not instances

Explanation: The `static` keyword is used to define a property or method that belongs to the class itself rather than to instances of the class.

54. How do you create an instance of a class in TypeScript?

- A) `let obj = new ClassName()`

- B) `let obj = create ClassName()`

- C) `let obj = ClassName()`

- D) `let obj = new ClassName`

- Answer: A) `let obj = new ClassName()`

Explanation: An instance of a class is created using the `new` keyword followed by the class name and parentheses.

55. How do you define a constructor in a TypeScript class?

- A) `constructor(parameters) {}`

- B) `constructor: parameters {}`

- C) `function constructor(parameters) {}`

- D) `constructor(parameters): returnType {}`

- Answer: A) `constructor(parameters) {}`

Explanation: A constructor in a TypeScript class is defined using the `constructor` keyword followed by parameters and the function body.

56. Which keyword is used to inherit a class in TypeScript?

- A) `extends`

- B) `implements`

- C) `inherits`

- D) `derives`

- Answer: A) `extends`

Explanation: The `extends` keyword is used to inherit a class in TypeScript.

57. How do you call a parent class's constructor in a derived class?

- A) `super(parameters)`

- B) `parent(parameters)`

- C) `base(parameters)`

-

D) `this(parameters)`

- Answer: A) `super(parameters)`

Explanation: The `super` keyword is used to call a parent class's constructor in a derived class.

58. How do you define a private property in a TypeScript class?

- A) `private property: type`

- B) `property private: type`

- C) `property: private type`

- D) `private: property type`

- Answer: A) `private property: type`

Explanation: A private property in a TypeScript class is defined using the `private` keyword before the property name and type.

59. How do you define a protected property in a TypeScript class?

- A) `protected property: type`

- B) `property protected: type`

- C) `property: protected type`

- D) `protected: property type`

- Answer: A) `protected property: type`

Explanation: A protected property in a TypeScript class is defined using the `protected` keyword before the property name and type.

60. How do you define a read-only property in a TypeScript class?

- A) `readonly property: type`

- B) `property readonly: type`

- C) `property: readonly type`

- D) `readonly: property type`

- Answer: A) `readonly property: type`

Explanation: A read-only property in a TypeScript class is defined using the `readonly` keyword before the property name and type.

### More TypeScript Features

61. What is the purpose of the `unknown` type in TypeScript?

- A) To denote any type of value

- B) To denote a value whose type is not known

- C) To denote a null or undefined value

- D) To denote a type-safe alternative to `any`

- Answer: D) To denote a type-safe alternative to `any`

Explanation: The `unknown` type is used as a type-safe alternative to `any`, ensuring that the value must be type-checked before being used.

62. How do you create a function that accepts a variable number of arguments in TypeScript?

- A) `function func(...args: type[])`

- B) `function func(args: type[])`

- C) `function func(args: type...)`

- D) `function func(type... args)`

- Answer: A) `function func(...args: type[])`

Explanation: To create a function that accepts a variable number of arguments, use the rest parameter syntax `function func(...args: type[])`.

63. Which of the following is a correct way to define a function type in TypeScript?

- A) `type FuncType = (param1: type, param2: type) => returnType;`

- B) `type FuncType = function(param1: type, param2: type): returnType;`

- C) `type FuncType = {param1: type, param2: type}: returnType;`

- D) `type FuncType = function(param1: type, param2: type) => returnType;`

- Answer: A) `type FuncType = (param1: type, param2: type) => returnType;`

Explanation: The correct syntax to define a function type is `type FuncType = (param1: type, param2: type) => returnType;`.

64. What does the `readonly` modifier do when applied to an array?

- A) Prevents adding new elements to the array

- B) Prevents modifying the array elements

- C) Prevents reassigning the array variable

- D) Prevents accessing the array elements

- Answer: B) Prevents modifying the array elements

Explanation: The `readonly` modifier applied to an array prevents modifying the elements of the array, making it immutable.

65. How do you define a union type in TypeScript?

- A) `type UnionType = type1 & type2`

- B) `type UnionType = type1 | type2`

- C) `type UnionType = type1 || type2`

- D) `type UnionType = type1 and type2`

- Answer: B) `type UnionType = type1 | type2`

Explanation: A union type is defined using the `|` operator, allowing a value to be one of several types.

66. Which of the following keywords is used to define a module in TypeScript?

- A) `module`

- B) `namespace`

- C) `package`

- D) `library`

- Answer: B) `namespace`

Explanation: The `namespace` keyword is used to define a module in TypeScript, providing a way to group related code.

67. What is the purpose of the `extends` keyword in TypeScript?

- A) To define a subclass that inherits properties and methods from a superclass

- B) To define an interface that inherits properties from another interface

- C) To specify type constraints in generics

- D) All of the above

- Answer: D) All of the above

Explanation: The `extends` keyword is used in multiple contexts, including defining subclasses, interface inheritance, and specifying type constraints in generics.

68. Which TypeScript feature allows defining a function or method that can operate on different data types?

- A) Abstract classes

- B) Union types

- C) Generics

- D) Interfaces

- Answer: C) Generics

Explanation: Generics allow defining a function or method that can operate on different data types, providing a way to create reusable and type-safe components.

69. How do you cast a variable to a specific type in TypeScript?

- A) `variable as type`

- B) `variable:type`

- C) `variable -> type`

- D) `variable is type`

- Answer: A) `variable as type`

Explanation: To cast a variable to a specific type, use the `as` keyword followed by the type, like `variable as type`.

70. What is the purpose of the `type` keyword in TypeScript?

- A) To declare a class

- B) To define a type alias

- C) To create an enum

- D) To define a module

- Answer: B) To define a type alias

Explanation: The `type` keyword is used to define a type alias, providing a way to give a new name to an existing type or a combination of types.

### TypeScript and Modules

71. How do you import all exported members from a module as a single object in TypeScript?

- A) `import \* as Module from 'module'`

- B) `import Module from 'module'`

- C) `import all from 'module'`

- D) `import module as \* from 'module'`

- Answer: A) `import \* as Module from 'module'`

Explanation: To import all exported members from a module as a single object, use the syntax `import \* as Module from 'module'`.

72. Which of the following is true about default exports in TypeScript?

- A) A module can have multiple default exports

- B) A module can have only one default export

- C) Default exports are not supported in TypeScript

- D) Default exports can only be functions

- Answer: B) A module can have only one default export

Explanation: A module can have only one default export, which can be a function, class, object, or any other value.

73. How do you import a default export from a module in TypeScript?

- A) `import DefaultExport from 'module'`

- B) `import { DefaultExport } from 'module'`

- C) `import \* as DefaultExport from 'module'`

- D) `import default as DefaultExport from 'module'`

- Answer: A) `import DefaultExport from 'module'`

Explanation: To import a default export from a module, use the syntax `import DefaultExport from 'module'`.

74. How do you export multiple members from a module in TypeScript?

- A) `export { member1, member2 }`

- B) `export all { member1, member2 }`

- C) `export members { member1, member2 }`

- D) `export \* { member1, member2 }`

- Answer: A) `export { member1, member2 }`

Explanation: To export multiple members from a module, use the syntax `export { member1, member2 }`.

75. How do you import specific named exports from a module in TypeScript?

- A) `import { member1, member2 } from 'module'`

- B) `import members { member1, member2 } from 'module'`

- C) `import { member1; member2 } from 'module'`

- D) `import { member1: member2 } from 'module'`

- Answer: A) `import { member1, member2

} from 'module'`

Explanation: To import specific named exports from a module, use the syntax `import { member1, member2 } from 'module'`.

76. How do you re-export a member from another module in TypeScript?

- A) `export { member } from 'module'`

- B) `export member from 'module'`

- C) `export default member from 'module'`

- D) `export \* as member from 'module'`

- Answer: A) `export { member } from 'module'`

Explanation: To re-export a member from another module, use the syntax `export { member } from 'module'`.

77. How do you export a member as the default export in TypeScript?

- A) `export default member`

- B) `export { default member }`

- C) `export { member as default }`

- D) `export member as default`

- Answer: A) `export default member`

Explanation: To export a member as the default export, use the syntax `export default member`.

78. How do you rename an imported member in TypeScript?

- A) `import { member as alias } from 'module'`

- B) `import alias from 'module'.member`

- C) `import { alias = member } from 'module'`

- D) `import member from 'module' as alias`

- Answer: A) `import { member as alias } from 'module'`

Explanation: To rename an imported member, use the syntax `import { member as alias } from 'module'`.

79. How do you import a module for its side effects only in TypeScript?

- A) `import 'module'`

- B) `import \* from 'module'`

- C) `import sideEffects from 'module'`

- D) `import module.sideEffects`

- Answer: A) `import 'module'`

Explanation: To import a module for its side effects only, use the syntax `import 'module'`.

80. Which keyword is used to export a member from a module in TypeScript?

- A) `export`

- B) `exported`

- C) `exports`

- D) `module.exports`

- Answer: A) `export`

Explanation: The `export` keyword is used to export a member (variable, function, class, etc.) from a module in TypeScript.

### TypeScript Types and Interfaces

81. What is the purpose of type guards in TypeScript?

- A) To enforce stricter type checks during runtime

- B) To allow type-safe access to properties of objects

- C) To narrow down the type within a conditional block

- D) To prevent type coercion

- Answer: C) To narrow down the type within a conditional block

Explanation: Type guards are used to narrow down the type of a variable within a conditional block, allowing for more precise type checks and operations.

82. How do you define a type alias in TypeScript?

- A) `type alias = definition;`

- B) `alias type = definition;`

- C) `define type alias = definition;`

- D) `alias = type definition;`

- Answer: A) `type alias = definition;`

Explanation: A type alias in TypeScript is defined using the `type` keyword followed by the alias name and the definition.

83. Which of the following is true about the `enum` type in TypeScript?

- A) It allows defining a set of named constants

- B) It enforces strict typing on string values

- C) It enables creating dynamic arrays

- D) It provides a way to define polymorphic behavior

- Answer: A) It allows defining a set of named constants

Explanation: The `enum` type in TypeScript is used to define a set of named constants, making the code more readable and maintainable.

84. How do you specify that a property can be of multiple types in TypeScript?

- A) `property: type1 or type2`

- B) `property: type1 & type2`

- C) `property: type1 | type2`

- D) `property: either(type1, type2)`

- Answer: C) `property: type1 | type2`

Explanation: To specify that a property can be of multiple types, use the union type syntax `property: type1 | type2`.

85. What is the purpose of the `abstract` keyword in TypeScript?

- A) To define a base class that cannot be instantiated directly

- B) To create a type alias

- C) To declare a function with no implementation

- D) To specify a default value for a property

- Answer: A) To define a base class that cannot be instantiated directly

Explanation: The `abstract` keyword is used to define a base class that cannot be instantiated directly but can be extended by other classes.

86. What is the purpose of the `readonly` modifier in TypeScript?

- A) To create a read-only variable

- B) To prevent a variable from being reassigned

- C) To create a constant variable

- D) To create a read-only property in an object or class

- Answer: D) To create a read-only property in an object or class

Explanation: The `readonly` modifier in TypeScript is used to make a property read-only, meaning it cannot be reassigned after the initial assignment.

87. What is the `unknown` type in TypeScript?

- A) A type-safe alternative to `any`

- B) A type that can be used in place of `void`

- C) A type used to define a union of all types

- D) A type that represents a null or undefined value

- Answer: A) A type-safe alternative to `any`

Explanation: The `unknown` type is a type-safe alternative to `any`, ensuring that the value must be type-checked before being used.

88. How do you create a tuple in TypeScript?

- A) `let tuple: [type1, type2]`

- B) `let tuple: {type1, type2}`

- C) `let tuple: (type1, type2)`

- D) `let tuple: [type1; type2]`

- Answer: A) `let tuple: [type1, type2]`

Explanation: A tuple in TypeScript is created using square brackets with the types of each element specified, like `let tuple: [type1, type2]`.

89. What is the purpose of the `never` type in TypeScript?

- A) To indicate that a function never returns

- B) To indicate that a function always returns a value

- C) To define an optional parameter

- D) To create a variable that can hold any type

- Answer: A) To indicate that a function never returns

Explanation: The `never` type is used to indicate that a function never returns, usually because it always throws an error or has an infinite loop.

90. How do you define a constant variable in TypeScript?

- A) `const variable: type = value`

- B) `let variable: type = value`

- C) `var variable: type = value`

- D) `constant variable: type = value`

- Answer: A) `const variable: type = value`

Explanation: A constant variable in TypeScript is defined using the `const` keyword, like `const variable: type = value`.

### TypeScript and OOP

91. Which keyword is used to prevent a class from being inherited in TypeScript?

- A) `sealed`

- B) `final`

- C) `static`

- D) TypeScript does not support this directly

- Answer: D) TypeScript does not support this directly

Explanation: TypeScript does not have a direct keyword to prevent a class from being inherited. However, you can prevent inheritance through design patterns and documentation.

92. What is method overloading in TypeScript?

- A) Defining multiple methods with the same name but different parameters

- B) Defining multiple methods with the same name and parameters

- C) Defining a method in both the superclass and subclass

- D) Defining a method that calls another method

- Answer: A) Defining multiple methods with the same name but different parameters

Explanation: Method overloading involves defining multiple methods with the same name but different parameters to provide different implementations based on the input arguments.

93. How do you declare a private constructor in TypeScript?

- A) `private constructor() {}`

- B) `constructor private() {}`

- C) `constructor() private {}`

- D) `private: constructor() {}`

- Answer: A) `private constructor() {}`

Explanation: A private constructor is declared using the `private` keyword before the `constructor` keyword, like `private constructor() {}`.

94. What is the purpose of the `

instanceof` operator in TypeScript?

- A) To check if an object is an instance of a specific class or constructor function

- B) To check if a variable is defined

- C) To cast an object to a specific type

- D) To check if a property exists on an object

- Answer: A) To check if an object is an instance of a specific class or constructor function

Explanation: The `instanceof` operator checks if an object is an instance of a specific class or constructor function, returning `true` or `false`.

95. How do you declare a readonly property in a TypeScript class?

- A) `readonly property: type;`

- B) `property readonly: type;`

- C) `property: readonly type;`

- D) `readonly: property type;`

- Answer: A) `readonly property: type;`

Explanation: A readonly property in a TypeScript class is declared using the `readonly` keyword followed by the property name and its type.

96. What is an abstract class in TypeScript?

- A) A class that cannot be instantiated directly and is meant to be subclassed

- B) A class that does not have any properties

- C) A class that can be instantiated but not extended

- D) A class that only has static methods

- Answer: A) A class that cannot be instantiated directly and is meant to be subclassed

Explanation: An abstract class is designed to be subclassed and cannot be instantiated directly. It can include abstract methods that must be implemented by subclasses.

97. How do you define a static method in a TypeScript class?

- A) `static methodName() {}`

- B) `methodName static() {}`

- C) `methodName() static {}`

- D) `static: methodName() {}`

- Answer: A) `static methodName() {}`

Explanation: A static method is defined using the `static` keyword before the method name, like `static methodName() {}`.

98. Which of the following is a correct way to implement inheritance in TypeScript?

- A) `class ChildClass extends ParentClass {}`

- B) `class ChildClass implements ParentClass {}`

- C) `class ChildClass inherits ParentClass {}`

- D) `class ChildClass extends from ParentClass {}`

- Answer: A) `class ChildClass extends ParentClass {}`

Explanation: Inheritance is implemented using the `extends` keyword, like `class ChildClass extends ParentClass {}`.

99. How do you define an interface with optional properties in TypeScript?

- A) `interface MyInterface { property?: type; }`

- B) `interface MyInterface { property: type?; }`

- C) `interface MyInterface { ?property: type; }`

- D) `interface MyInterface { property: optional type; }`

- Answer: A) `interface MyInterface { property?: type; }`

Explanation: Optional properties in an interface are defined using the `?` symbol, like `property?: type`.

100. What is the purpose of a constructor in a TypeScript class?

- A) To initialize the class's properties when an instance is created

- B) To define methods in a class

- C) To implement interfaces

- D) To enforce type constraints

- Answer: A) To initialize the class's properties when an instance is created

Explanation: The constructor is a special method used to initialize the properties of a class when an instance is created.