TS: Exercises for Functions

Depicted Candela

September 2, 2024

Function Types and Signatures

Exercise 1: Function Type Expressions

1. Define a function type for a function that calculates the distance between two points in a 2D space (each point has x and y coordinates).
2. Create a function that matches the above type and computes the distance.

Exercise 2: Call and Construct Signatures

1. Define a callable object that can be used to calculate the perimeter of a rectangle given width and height. The object should also have a method scale(factor: number) to scale the dimensions of the rectangle. Include both call and construct signatures.

Exercise 3: Declaring this in Functions

1. Write a function for a simple geometry object that calculates the area of a triangle. Ensure the function can be used as a method and specify the type of `this` to ensure type safety when invoked.

Generics in TypeScript Functions

Exercise 4: Writing Generic Functions

1. Create a generic function that accepts an array of geometric shapes (e.g., rectangles, circles) and returns the shape with the maximum area. Use generics to ensure type safety for different shape types.

Exercise 5: Type Inference and Constraints

1. Define a generic function to merge properties of two objects: one representing a geometric shape and another containing additional metadata (e.g., color, name). Use constraints to ensure the objects have specific properties (e.g., shapes must have an area method).

Exercise 6: Guidelines for Writing Good Generic Functions

1. Refactor the previous exercise's generic function to reduce the number of type parameters and ensure that each type parameter appears at least twice in the function signature.

Function Overloads and Usage

Exercise 7: Writing Function Overloads

1. Create function overloads for a function named `translate` that moves a geometric shape by a specified distance. It should handle:
    - Moving by a fixed distance in both x and y directions.
    - Moving by a distance object with dx and dy properties.

Exercise 8: Overload Signatures vs. Implementation Signatures

1. Implement the `translate` function with correct overloads and an implementation signature that handles both overload cases correctly.

Exercise 9: Additional Guidelines for Overloads

1. Modify the `translate` function to use union types instead of overloads where appropriate, and compare the results in terms of simplicity and readability.

Optional Parameters and Callback Functions

Exercise 10: Optional Parameters

1. Write a function that calculates the area of a polygon with optional parameters for units (e.g., square meters, square kilometers). The default should be square meters.

Exercise 11: Optional Parameters in Callbacks

1. Define a function that takes a callback to compute the centroid of a set of points. The callback should optionally accept an error parameter if the input set is empty or invalid.

Interactive Code Examples and Practical Application

Exercise 12: Interactive Examples Using TypeScript Playground

1. Using the TypeScript Playground, create an interactive example of a function that calculates the convex hull of a set of points. Experiment with different function types and generics to ensure type safety and flexibility.

Exercise 13: Real-World Scenarios

1. Implement a small API for a geometric application that includes functions for adding shapes, computing areas, and querying shapes by type (e.g., all circles). Ensure the functions are typed correctly and handle common edge cases.

Advanced TypeScript Function Topics

Exercise 14: Syntax Differences and Advanced Requirements

1. Compare and contrast two functions: one using a function expression and another using an arrow function. Discuss the implications of this in each case within the context of a geometry application.

Function Type Expressions

Exercise 15: Basic Syntax and Examples

1. Define a function type for a function that calculates the angle between two vectors in 2D space. Implement a function matching this type.

Call Signatures

Exercise 16: Defining Call Signatures

1. Create a call signature for a geometry object that represents a vector with properties and a method `magnitude()` that returns its length.

Construct Signatures

Exercise 17: Writing Construct Signatures

1. Define a construct signature for a class that represents a geometric shape that can be instantiated with `new` to create shapes like circles, rectangles, or polygons.

Generic Functions

Exercise 18: Using Type Parameters in Function Signatures

1. Write a generic function that takes a shape and a transformation function, and returns the transformed shape. Ensure type safety through appropriate use of type parameters.

Optional Parameters

Exercise 19: Handling Optional Parameters in Callbacks

1. Define a callback function type that computes a geometric transformation (e.g., scaling, rotating) and can optionally handle errors related to invalid transformations.

Function Overloads

Exercise 20: Creating Functions with Multiple Signatures

1. Implement a function `measure()` that accepts either a single geometric shape to measure its perimeter or multiple shapes to measure the total perimeter.

Declaring this in a Function

Exercise 21: Specifying the Type of this in a Function Body

1. Write a method for a geometry object that uses `this` to refer to the current object, ensuring the type of `this` is correctly specified for type safety.