***advanced TypeScript examples,***

***including classes, interfaces, functions, and decorators***

**Classes:**

TypeScript

class Person {

constructor(public name: string, public age: number) {}

greet(): string {

return `Hello, my name is ${this.name} and I am ${this.age} years old.`;

}

}

const john = new Person("John Doe", 30);

console.log(john.greet());

//EXPLANATION:

class Person { ... }: This line declares a new class named Person

constructor(public name: string, public age: number) {}:

* The constructor is a special method that is called when a new object of the class is created. It's used to initialize the object's properties.
* public name: string: This line defines a property named name that will hold a string value. The public keyword makes this property accessible from outside the class.
* public age: number: Similar to name, this defines a property named age to hold a number value and makes it publicly accessible.

**Method:**

* greet(): string { ... }: This line defines a method named greet that returns a string value.
  + greet(): This is the name of the method that can be called on an object of the Person class.
  + (): string: This part specifies that the method doesn't take any arguments (()) and returns a string value (string).
  + return ...: Inside the method body, you have a string template literal. String template literals allow you to embed expressions within strings using backticks (``).
    - Hello, my name is ${this.name} and I am ${this.age} years old.: This string uses template literals to insert the values of the name and age properties (accessed using this) into the string

**Object Creation and Method Call:**

* const john = new Person("John Doe", 30);: This line creates a new object of the Person class.
  + new Person(...): The new keyword is used to create a new instance of a class.
  + "John Doe", 30: These values are passed as arguments to the constructor, initializing the name and age properties of the newly created object.
* console.log(john.greet());: This line calls the greet method on the john object.
  + john.greet(): We access the greet method using dot notation on the john object, which is an instance of the Person class.
  + console.log(...): This prints the returned value from the greet method to the console.

**Interfaces:**

TypeScript

interface Shape {

area(): number;

}

class Circle implements Shape {

constructor(public radius: number) {}

area(): number {

return Math.PI \* this.radius  \* 2;

}

}

class Rectangle implements Shape {

constructor(public width: number, public height: number) {}

area(): number {

return this.width \* this.height;

}

}

const circle = new Circle(5);

const rectangle = new Rectangle(4, 3);

console.log(circle.area()); // Output: 78.53981633974483

console.log(rectangle.area()); // Output: 12

// explanation

**Shape interface:** This interface defines a contract or blueprint for any class that wants to be considered a "shape." It specifies that any class implementing this interface must have a method named area() that returns a number. This ensures that all shapes have a consistent way to calculate their area.

**Classes:**

* **Circle class:** This class implements the Shape interface, meaning it adheres to the contract defined by the interface. It has a radius property and an area() method that calculates the area of a circle using the formula π \* radius \* radius.
* **Rectangle class:** This class also implements the Shape interface, providing a width and height property. Its area() method calculates the area of a rectangle using the formula width \* height.

**Object Creation and Method Calls:**

* **const circle = new Circle(5);:** This line creates a new object of the Circle class with a radius of 5.
* **const rectangle = new Rectangle(4, 3);:** This line creates a new object of the Rectangle class with a width of 4 and a height of 3.
* **console.log(circle.area());:** This line calls the area() method on the circle object, which calculates and prints the area of the circle (78.53981633974483).
* **console.log(rectangle.area());:** This line calls the area() method on the rectangle object, which calculates and prints the area of the rectangle (12).

**Functions:**

TypeScript

function greet(name: string): string {

return `Hello, ${name}!`;

}

const greetArrow = (name: string) => `Hello, ${name}!`;

console.log(greet("Alice"));

console.log(greetArrow("Bob"));

// explanation

* **greet(name: string): string:** This line defines a function named greet that takes a string parameter named name and returns a string. The : string part indicates that the function's return type is a string.
* **returnHello, ${name}!;:** This line is the body of the function, and it returns a string that includes the greeting "Hello, " followed by the value of the name parameter. The ${name} syntax is a template literal, which allows you to embed expressions within strings.
* **const greetArrow = (name: string) =>Hello, ${name}!;:** This line defines a function using arrow function syntax. Arrow functions are a concise way to write functions in JavaScript and TypeScript. The => symbol separates the function's parameters from its body. The const keyword declares the function as a constant, meaning it cannot be reassigned.
* **console.log(greet("Alice"));:** This line calls the greet function with the argument "Alice" and logs the returned value to the console. The output will be "Hello, Alice!".
* **console.log(greetArrow("Bob"));:** This line calls the greetArrow function with the argument "Bob" and logs the returned value to the console. The output will be "Hello, Bob!".

**Decorators:**

TypeScript

function log(target: any, propertyKey: string, descriptor: PropertyDescriptor) {

const originalMethod = descriptor.value;

descriptor.value = function(...args: any[]) {

console.log(`Calling

${propertyKey} with arguments: ${args}`);

const result = originalMethod.apply(this, args);

console.log(`Result: ${result}`);

return result;

};

return descriptor;

}

class Calculator {

@log

add(a: number, b: number): number {

return a + b;

}

}

const calculator = new Calculator();calculator.add(2,3);

**// explanation**

**Decorators:**

* **log decorator:** This function is a decorator, which is a special type of function that can be applied to classes, properties, or methods using the @ symbol. Decorators are used to modify the behavior of the decorated element.
* **target:** This parameter represents the class or object that the decorator is applied to.
* **propertyKey:** This parameter represents the name of the property or method being decorated.
* **descriptor:** This parameter represents the property descriptor object, which contains information about the property or method, such as its value, enumerability, and configurability.
* **originalMethod:** This variable stores the original value of the decorated method.
* **descriptor.value = function(...args: any[]) { ... }:** This line replaces the original method with a new function that logs the function call and its arguments before and after executing the original method.
* **return descriptor;:** This line returns the modified property descriptor, which will be used by TypeScript to apply the decorator's changes.

**Class and Method:**

* **Calculator class:** This class defines a calculator object with an add method.
* **@log:** This line applies the log decorator to the add method. This means that the log function will be called with the Calculator class, the add property key, and the add method's descriptor as arguments.
* **add(a: number, b: number): number:** This is the add method, which takes two numbers as input and returns their sum.

**Execution:**

* **const calculator = new Calculator();:** This line creates a new instance of the Calculator class.
* **calculator.add(2, 3);:** This line calls the add method on the calculator object with the arguments 2 and 3. When the add method is called, the log decorator intercepts the call and logs the following:
  + "Calling add with arguments: 2,3"
  + "Result: 5"