# Programming and Design Questions Answers

## 1. newWithoutNew() Function

function newWithoutNew(constructor, ..args) {

// Create a new object that inherits from the constructor's prototype

const obj = Object.create(constructor.prototype);

// Call the constructor function with the new object as context

const result = constructor.apply(obj, args);

// Return the new object or the result if the constructor returns an object

return (typeof result === 'object' && result !== null) ? result : obj;

}

// Example usage

function C(a, b) {

this.a = a;

this.b = b;

}

C.prototype.f = function(x) { return this.a \* x + this.b; };

const c32 = newWithoutNew(C, 3, 2);

console.log(c32); // C { a: 3, b: 2 }

console.log(c32.f(4)); // 14

console.log(c32.constructor); // [Function: C]

console.log(c32 instanceof C); // true

## 2. Setting a Property to be Non-Writable

You can make a property non-writable using Object.defineProperty() or by using a getter without a setter. Here are two methods:

### Method 1: Object.defineProperty

function ClassWithConstProp1() {

Object.defineProperty(this, 'prop', {

value: 42,

writable: false, // Makes the property non-writable

configurable: true

});

}

const c1 = new ClassWithConstProp1();

console.log(c1.prop); // 42

c1.prop = 33; // This assignment will be ignored

console.log(c1.prop); // 42

### Method 2: Using a Getter

function ClassWithConstProp2() {

let \_prop = 42; // Private variable

Object.defineProperty(this, 'prop', {

get: function() {

return \_prop;

},

set: function(value) {

// No setter, making it effectively non-writable

}

});

}

const c2 = new ClassWithConstProp2();

console.log(c2.prop); // 42

c2.prop = 33; // This assignment will be ignored

console.log(c2.prop); // 42

## 3. HTTP DELETE Request and Idempotency

In RESTful services, the DELETE method is idempotent, meaning that multiple identical requests should have the same effect as a single request.

* **Option 1:** Return 204 NO CONTENT - This approach emphasizes the idempotent nature of DELETE.
* **Option 2:** Return 404 NOT FOUND - This indicates that the resource no longer exists.

**Recommendation:** The first approach (returning 204 NO CONTENT) is generally preferred because it aligns with the principle of idempotency.

## 4. REST API Design for Tree Census

- GET /trees

Description: Retrieve a list of all trees.

Response: JSON array of tree objects.

Cache-Control: max-age=300

- GET /trees/{id}

Description: Retrieve a specific tree by ID.

Response: JSON object of the tree.

Error: 404 NOT FOUND if the tree does not exist.

- POST /trees

Description: Add a new tree.

Request Body: JSON object with tree attributes.

Response: JSON object of the created tree with ID.

Error: 400 BAD REQUEST for invalid data.

- PATCH /trees/{id}

Description: Update an existing tree.

Request Body: JSON object with attributes to update.

Response: JSON object of the updated tree.

Error: 404 NOT FOUND if the tree does not exist.

- DELETE /trees/{id}

Description: Remove a tree by ID.

Response: 204 NO CONTENT.

Error: 404 NOT FOUND if the tree does not exist.

## 5. DRY Principle in Express.js Server

async function getResource(req, res, next) {

try {

const resource = await getResourceFromDb(req.query.id);

if (!resource) {

return res.status(404).send('Resource not found');

}

req.resource = resource; // Attach resource to request object

next(); // Proceed to the next handler

} catch (error) {

res.status(500).send('Server error');

}

}

app.get('/resource', getResource, (req, res) => {

res.json(req.resource);

});

app.patch('/resource', getResource, async (req, res) => {

// Update logic here using req.resource

res.json(updatedResource);

});

app.delete('/resource', getResource, async (req, res) => {

// Delete logic here using req.resource

res.status(204).send();

});