Postman Problem Set 2 Solutions

Useful examples of test scripts can be found here:
https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-examples/

1 Response Validation

Problem 1.1: Ensure response has a 'Content-Type' header of 'application/json'.

```
pm.test("Content-Type is application/json", function ()
   {
    pm.response.to.have.header("Content-Type", "
       application/json");
});
```

Problem 1.2: Verify that response JSON has a specific nested structure.

```
pm.test("Response contains user details", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.user).to.have.all.keys("id", "
       email", "name");
});
```

Problem 1.3: Ensure 'items' array is not empty and contains specific data.

```
pm.test("Items array is not empty", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.items).to.have.members([1, 2, 3])
       ;  // Example values
```

```
});
```

2 Handling Cookies

Problem 2.1: Check if a specific cookie exists in the response.

```
pm.test("Session cookie exists", function () {
    pm.expect(pm.cookies.get("sessionId")).to.not.be.
       null;
});
```

Problem 2.2: Validate cookie value format (alphanumeric).

```
pm.test("Session cookie is alphanumeric", function () {
    pm.expect(pm.cookies.get("sessionId")).to.match(/^[a
       -zA-Z0-9]+$/);
});
```

Problem 2.3: Ensure specific cookies are set after login.

```
pm.test("Cookies are set correctly after login",
   function () {
    pm.expect(pm.cookies.has("sessionId")).to.be.true;
    pm.expect(pm.cookies.has("authToken")).to.be.true;
});
```

3 Assertions and Loops

Problem 3.1: Verify each item in the array has a unique 'id'.

```
pm.test("Each item has a unique id", function () {
    var jsonData = pm.response.json();
    var ids = jsonData.items.map(item => item.id);
    pm.expect(new Set(ids).size).to.equal(ids.length);
});
```

Problem 3.2: Assert that all user objects have required properties using deep comparison.

```
pm.test("Each user has required fields", function () {
    var jsonData = pm.response.json();
    jsonData.users.forEach(user => {
        pm.expect(user).to.have.all.keys("id", "email",
           "name", "created_at");
    });
});
```

Problem 3.3: Validate response time is within a certain range and contains a 'status' key.

```
pm.test("Response time and status check", function () {
    pm.expect(pm.response.responseTime).to.be.below
       (1000);  // Response time should be below 1000ms
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("status").that.
       equals("success");
});
```

4 Nested Loops and Conditional Logic

Problem 4.1: Check if each item has a valid 'price' within a specified range.

```
pm.test("Each item has a valid price", function () {
    var jsonData = pm.response.json();
    jsonData.items.forEach(item => {
        pm.expect(item.price).to.be.within(10, 1000);
           // Price should be between 10 and 1000
    });
});
```

Problem 4.2: Validate at least one item in the array has 'status' as 'active'.

```
pm.test("At least one item is active", function () {
    var jsonData = pm.response.json();
    const activeItem = jsonData.items.some(item => item.
       status === "active");
    pm.expect(activeItem).to.be.true;
});
```

Problem 4.3: Validate multiple assertions on a user profile.

```
pm.test("User profile is valid", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.user.name).to.equal("Jane Doe");
    pm.expect(jsonData.user.email).to.match(/^\S+@\S+\.\
       S+$/);  // Valid email format
    pm.expect(jsonData.user.created_at).to.be.a('string
       ');  // Created date is a string
});
```

5 Others

Problem 5.1: Validate dynamic authentication token and test for authorization. Using the token from the login API, make a request to a protected resource and verify that the response status is 200 and the token is valid.

```
pm.test("Authorization is successful with valid token",
   function () {
    var token = pm.environment.get("authToken");
    pm.response.to.have.status(200);
    pm.expect(token).to.match(/^[a-zA-Z0-9]+$/);  //
       Token should be alphanumeric
});
```

Problem 5.2: Ensure all users have unique email addresses.

```
pm.test("All users have unique emails", function () {
    var jsonData = pm.response.json();
    var emails = jsonData.users.map(user => user.email);
    var uniqueEmails = new Set(emails);
    pm.expect(uniqueEmails.size).to.equal(emails.length)
       ;
});
```

Problem 5.3: Check that the response includes a 'X-RateLimit-Remaining' header and it's a positive number.

```
pm.test("X-RateLimit-Remaining header is present and
   positive", function () {
    pm.response.to.have.header("X-RateLimit-Remaining");
    var remaining = pm.response.headers.get("X-RateLimit
       -Remaining");
    pm.expect(Number(remaining)).to.be.above(0);
});
```