# API Integration Task: ING Open Banking – Payment Request

**Objective**

You are required to implement a feature that interacts with the **ING Open Banking API** to create a **payment request**. This task will test your ability to work with external APIs, handle authentication, and structure a proper request.

**Instructions**

1. **Read the ING Open Banking API documentation** for **creating a payment request**:  
   [ING Developer Portal - Premium API](https://developer.ing.com/openbanking/resources/get-started/premium#sandbox-and-production)
2. **Develop a PHP class or function** that:
   * Authenticates with the ING API (use OAuth2 and mTLS).
   * Sends a request to create a **payment request**.
   * Handles API responses and errors properly.
3. **Ensure the solution is structured and maintainable**, using clean coding practices.
4. You can optionally use any framework that you want or just create the necessary PHP files.

**Requirements**

* Use PHP (you may use cURL or Guzzle for HTTP requests).
* The function should accept the following parameters:
  + amount: The amount to request (e.g., 50.00).
  + currency: The currency for the payment (default: EUR).
  + description: A description for the payment.
  + callback\_url: The URL where the user should be redirected after payment.
* The function should:
  + Retrieve an **access token** from ING's OAuth2 authentication system.
  + Use the token to **send a payment request**.
  + Return the **payment request URL** for the user to complete the transaction.

**Example Usage**

$paymentRequest = INGPayment::createPaymentRequest(50.00, "EUR", "Order #12345", "https://yourdomain.com/payment/callback");  
echo "Payment request created: " . $paymentRequest;

**Expected Output**

The function should return a **valid URL** to initiate the payment, such as:

https://pay.ing.com/redirect/abcd1234

**Evaluation Criteria**

* **Correct API implementation**: Properly follow the ING Open Banking API requirements.
* **Security & error handling**: Handle authentication securely and manage API errors gracefully.
* **Code quality**: Use clean, structured, and reusable code.

**Bonus (Optional)**

* Implement logging for API requests and responses.
* Use Request Message Signing option from ING API.

**Submission**

* Provide a **fully functional PHP script** or class.
* Include **instructions on how to run it** (any required API credentials, setup steps).
* Explain the **key challenges faced** and **how you solved them**.

# Coding Test: Code Analysis & Optimization

**Objective**

Your task is to analyze and optimize specific methods within the provided TestTools.php file. The goal is to identify inefficiencies, propose improvements, and ensure that the optimized code produces the same results as the original implementation.

**Instructions**

1. Carefully review the methods **evalFormula()** and **generateRandomValues()** within TestTools.php.
2. Identify any performance bottlenecks, redundant operations, or unnecessary complexity.
3. Rewrite these methods to improve efficiency, maintainability, and readability.
4. Ensure that the optimized code **produces the same output** as the original when executed with the given test cases.

**Evaluation Criteria**

* **Code efficiency**: Reduce unnecessary operations while maintaining correctness.
* **Readability & maintainability**: Use clean coding practices and meaningful variable names.
* **Functionality preservation**: The optimized version must return the same expected results.

**Test Cases**

Your optimizations must pass the following tests:

**Formula Processing**

$macro1 = '[MACRO]date(Y-m-d H:i:s)[#MACRO]';  
$macro2 = '[MACRO]division(42|2|3)[#MACRO]';  
$macro3 = '[MACRO]number\_format([MACRO]multiply(1000.817232|5|100)[#MACRO]|2|,|.)[#MACRO]';  
$output1 = TestTools::getFormulaFunctions($macro1);  
// Expected result: "current date" (formatted as Y-m-d H:i:s)

$output2 = TestTools::getFormulaFunctions($macro2);  
// Expected result: "7"

$output3 = TestTools::getFormulaFunctions($macro3);  
// Expected result: "500.408,62"

**Random Value Generation**

$output4 = TestTools::generateRandomValues('-red#-blue#-green#-1', 4, 'LIST');  
// Expected result: A random string like "-blue-blue-1-red"

**Hints for Optimization**

* **evalFormula()**: Check for unnecessary operations, redundant string manipulations, and inefficient loops.
* **generateRandomValues()**: Look for redundant condition checks, excessive type conversions, and unnecessary logic in character selection.

**Submission**

* Provide a **list of improvements** made to each method.
* Submit the **optimized code** for both methods.
* Include a brief explanation of your approach and how it improves the code.

Good luck!