Instructions:

Run atleast 20 of the prompts in this document, each prompt being run in parallel in tools like ChatGPT, Google Gemini, Claud.ai/chat, Perplexity and identify which tool is consistently giving you the best outputs.

Prompts for testing Reasoning capabilities of LLMs:

Go to

https://www.manhattanreview.com/gmat-practice-questions/ and

https://www.manhattanreview.com/gre-practice-questions/

Take Qs from various sections like Reading Comprehension, Critical Reasoning, etc and use above tools to get the answers, compare with answers of the free test Qs.

Prompts for testing Machine Learning development related capabilities of LLMs:

1. Create the image of a stop road sign. Next, apply a kNN classifier to help the car recognize this sign and output the sign caption. Also show the code used to create the output.
2. I need to create a binary classification model for detecting fraudulent transactions (transactions not done by customer but by a fraudster, on the card of the customer) for credit card Industry. Please create a synthetic dataset of 1000 credit card transactions with 80% of the transactions labelled as “not fraud” and other 20% labelled as “fraud”. Next, build the model by training it with the above synthetic data created and show the code used to create the model.

Other Technical Prompt Engineering scenarios:

- Create a Flask API for managing a to-do list. The API should have endpoints to add a new task, retrieve all tasks, update a task status, delete a task.
- I want to write a test case performing the following steps:

1. Open the browser to https://katalon-demo-cura.herokuapp.com/

2. Click the make appointment button named 'Page_CuraHomepage/btn_MakeAppointment'

3. Fill username in the 'Page_Login/txt_Username' object with the value in the 'Username' variable

4. Fill the password in the Page_Login/txt_Password' object with the value in the 'Password' variable

5. Verify that the appointment div 'Page_CuraAppointment/div_Appointment' exists within 10 seconds.

6. Close the browser

Few-Shot Prompting:

Examples generated during discussion:

Generate Technical Architecture diagram for creating a on-demand video streaming application like Youtube using AWS Cloud and AWS services.

Please review this code: "<enter code>". Please provide granular details of which parts of the code has any issues that are highlighted in the review. Next, specify the parameters and guidelines that you used to do the code review.

Other Examples:

Example 1: Classifying Emails

Prompt:

I have examples of classifying emails into "Work" or "Personal". Use these examples to classify the next email.

Example 1:

Input: "Meeting with team at 10 AM"

Output: Work

Example 2:

Input: "Dinner with friends at 8 PM"

Output: Personal

Now, classify the following email:

Input: "Submit the monthly report by tomorrow"

Output:

Example 2: Identifying Sentiments

Prompt:

I have examples of identifying sentiments in customer feedback. Use these examples to determine the sentiment of the new feedback.

Example 1:

Input: "The product is amazing!"

Output: Positive

Example 2:

Input: "I am disappointed with the delivery."

Output: Negative

Input: "The packaging could have been better."

Output: Neutral

Now, identify the sentiment for this feedback:

Input: "Customer service was very helpful."

Output:

Example 3: Math Problem Solving

Prompt:

Solve the math problems as demonstrated below.

Example 1:

Input: "What is the square root of 16?"

Output: 4

Example 2:

Input: "What is 5 times 7?"

Output: 35

Now, solve the following problem:

Input: "What is 12 divided by 4?"

Output:

Example 4: Python – Generating an API Endpoint for User Authentication

Prompt:

Below are two examples of Python FastAPI endpoints for handling user-related operations. Use these examples to create python code for a new endpoint for resetting a user's password. The POST endpoint `/reset-password` will accept `email` and `new_password`, verify the email exists, update the password, and return a success message. Please add proper documentation to each line of code as the person who will use this code is a layman in coding.

Example 1:

```
@app.post("/register")
def register_user(user: User):
    # Logic to register a user
    return {"message": "User registered successfully"}
```

Example 2:

```
@app.post("/login")
def login_user(user: User):
    # Logic to authenticate a user
    return {"token": "abcdef123456"}
```

Example 5: Java – Generating REST API Endpoints for Product Management ***(Create an improved version of the prompt - try few combinations)***

Prompt:

Use the examples provided to generate a new API endpoint for deleting a product in Java using Spring Boot:

Example 1:

```
@PostMapping("/add-product")
public ResponseEntity<String> addProduct(@RequestBody Product product) {
   // Logic to add product
   return ResponseEntity.ok("Product added successfully");
}
```

Example 2:

```
@GetMapping("/get-product/{id}")
public ResponseEntity<Product> getProduct(@PathVariable int id) {
   // Logic to retrieve product by ID
   return ResponseEntity.ok(product);
}
```

New task:

Create a `@DeleteMapping` endpoint `/delete-product/{id}` to delete a product by ID. Ensure appropriate error handling if the product ID is not found.

Example 6: JavaScript – Creating Express.js Routes for Task Management

(Create an improved version of the prompt - try few combinations)

Prompt:

Based on the examples below, create a new Express.js route for updating a task:

Example 1:

```
app.post("/create-task", (req, res) => {
```

```
  const { title, description } = req.body;
  // Logic to create a task
  res.send("Task created successfully");
});
```

Example 2:

```
app.get("/tasks", (req, res) => {
  // Logic to fetch all tasks
  res.json(tasks);
});
```

New task:

Create a PUT route `/update-task/:id` that accepts `id` as a parameter and updates the task's title and description based on the request body. Ensure proper validation and error handling if the task ID does not exist.

Prompt Chaining:

Example 1: Converting Natural Language to Code (Python)

Prompt:

Task: Write a Python function that calculates the factorial of a number.

Step 1: Identify the input and output.

Input: A single integer (e.g., 5).

Output: The factorial of the number (e.g., 120).

Step 2: Write a function to compute the factorial using recursion.

Step 3: Test the function with edge cases like 0 and 1.

Now, complete the task step by step:

Example 2: Building a REST API (Java)

Prompt:

Task: Create a Java REST API endpoint for a "Todo List" application.

Step 1: Define a "Todo" object with fields like `id`, `task`, and `status`.

Step 2: Implement the `GET /todos` endpoint to return all todos.

Step 3: Implement the `POST /todos` endpoint to add a new todo.

Now, follow these steps to complete the task.

Example 3: Parsing JSON Data (JavaScript)

Prompt:

Task: Parse a JSON string in JavaScript and extract a specific value.

Step 1: Take a JSON string as input. Example: '{"name": "John", "age": 30}'

Step 2: Parse the JSON string to a JavaScript object.

Step 3: Extract the value of the "name" field.

Complete the task following these steps.

Chain of Thought:

Example 1: Debugging Python Code

Prompt:

Debug the following Python code step-by-step to identify and fix the error:

```
def calculate_average(nums):
    total = 0
    for num in nums:
        total += num
    average = total / len(nums)
    return average
```

Input: [10, 20, 0]

Step 1: Identify the inputs, outputs, and expected behavior.

Step 2: Check for potential edge cases (e.g., empty lists).

Step 3: Run the code and analyze any exceptions or incorrect results.

Step 4: Fix the identified issue and re-run the code.

Complete this task by following the steps provided.

Example 2: Creating an ETL Pipeline (Java)

Prompt:

Create an ETL (Extract, Transform, Load) pipeline in Java to process customer data from a database.

Step 1: Extract data from the database using JDBC.

Step 2: Transform the data by standardizing the names (capitalize first letter of each name).

Step 3: Load the transformed data into another database table.

Write a function for each step and then chain them together in a main program. Provide explanations for the steps as you code.

Example 3: JavaScript Workflow for API Testing

Prompt:

Write a multi-step JavaScript workflow to test an API endpoint.

Step 1: Use the `fetch` API to send a POST request to `https://api.example.com/login` with a username and password.

Step 2: Parse the response to retrieve a token.

Step 3: Use the token to send a GET request to `https://api.example.com/user-details`.

Step 4: Validate the status codes and the presence of required fields in both responses.

Implement each step in a modular function and chain them together in the workflow. Provide error handling for each step.

Negative Prompts:

Example 1: Avoiding Unnecessary Comments in Generated Python Code

Prompt:

Generate Python code for a function that calculates the Fibonacci sequence up to `n`.

Constraints:

- Avoid irrelevant comments like "This is a loop" or "Here we add two numbers."
- Focus only on comments that explain the logic of the algorithm or any edge cases handled.

Example of acceptable comments:

- "This handles the case where n is less than 2."
- "Using dynamic programming for efficiency."

Now generate the code based on these constraints.

Example 2: Avoiding Non-Essential Code in Java

Prompt:

Refactor the following Java code for a calculator program.

Constraints:

- Do not include unnecessary getters/setters for fields that are never used outside the class.
- Avoid redundant methods or overly verbose syntax.
- Ensure concise and meaningful variable names.

Example Input Code:

```
public class Calculator {
    private int result;
```

```
    public void setResult(int result) { this.result = result; }
    public int getResult() { return this.result; }
    public int add(int a, int b) {
        return a + b;
    }
}
```

Refactor the code while adhering to these constraints.

Example 3: Filtering Out Irrelevant Outputs in JavaScript

Prompt:

Write a JavaScript function to sort a list of employee objects by their salary in descending order.

Constraints:

- Avoid unnecessary outputs, such as logs of intermediate steps unless explicitly requested.
- Ensure the function only returns the sorted array.
- Provide meaningful names for variables and functions.

Example Input:

```
const employees = [
    { name: "Alice", salary: 60000 },
    { name: "Bob", salary: 75000 },
    { name: "Charlie", salary: 50000 }
];
```

Example Output:

```
[
    { name: "Bob", salary: 75000 },
```

```
  { name: "Alice", salary: 60000 },
  { name: "Charlie", salary: 50000 }
]
```

Write the function adhering to these constraints.

Writing Code:

Example 1: Python – Palindrome Check

Prompt:

Write a Python function to check if a given string is a palindrome.

The function should:

- Take a single string as input.

- Return `True` if the string is a palindrome, otherwise `False`.

Example:

Input: "radar"

Output: True

Example 2: Java – Prime Number Check

Prompt:

Write a Java program to check if a given number is a prime number.

The program should:

- Take an integer as input.

- Return `true` if the number is prime, otherwise `false`.

Example:

Input: 7

Output: true

Example 3: JavaScript – Sorting an Array

Prompt:

Write a JavaScript function to sort an array of integers in ascending order.

The function should:

- Take an array of integers as input.

- Return the sorted array.

Example:

Input: [5, 2, 9, 1]

Output: [1, 2, 5, 9]

Example 4: Java – Building a Simple Banking Application

Prompt:

Write a Java program to manage a simple banking application.

Requirements:

- Create a `BankAccount` class with fields: `accountNumber`, `accountHolder`, `balance`.

- Implement methods for:

  - `deposit(double amount)`: Add amount to the balance.

  - `withdraw(double amount)`: Deduct amount from the balance if sufficient funds exist.

- `printAccountDetails()`: Print account information.

Example:

Input:

BankAccount account = new BankAccount(12345, "Alice", 1000.0);

account.deposit(500.0);

account.withdraw(300.0);

account.printAccountDetails();

Output:

Account Number: 12345

Account Holder: Alice

Balance: 1200.0

Example 5: JavaScript – Employee Management System (Frontend Functionality)

Prompt:

Write a JavaScript program to manage employees in a company.

Requirements:

- Create a function `addEmployee` to add a new employee (with `id`, `name`, and `department`) to an array.

- Create a function `findEmployeeById` to find and return an employee object by `id`.

- Create a function `listEmployeesByDepartment` to return a list of employees filtered by department.

Example Input:

addEmployee({ id: 1, name: "John", department: "HR" });

addEmployee({ id: 2, name: "Jane", department: "IT" });

listEmployeesByDepartment("IT");

Example Output: [{ id: 2, name: "Jane", department: "IT" }]

Explaining Code:

Example 1: Python – Fibonacci Series

Prompt:

Explain the following Python code that generates a Fibonacci series:

```
def fibonacci(n):
    series = [0, 1]
    for i in range(2, n):
        series.append(series[-1] + series[-2])
    return series[:n]
```

The explanation should include:

- Purpose of the code.
- Functionality of each line.
- Example inputs and outputs.

Example 2: Java – Reversing a String

Prompt:

Explain the following Java code to reverse a string:

```
public class ReverseString {
    public static void main(String[] args) {
        String str = "Hello";
        String reversed = new StringBuilder(str).reverse().toString();
        System.out.println(reversed);
```

```
    }
}
```

The explanation should include:

- Purpose of the program.
- Explanation of each method used.
- Example input and output.

Example 3: JavaScript – Sum of Array Elements

Prompt:

Explain the following JavaScript code to calculate the sum of array elements:

```
const sumArray = (arr) => arr.reduce((acc, num) => acc + num, 0);
console.log(sumArray([1, 2, 3, 4]));
```

The explanation should include:

- Purpose of the function.
- Functionality of each method used.
- Example inputs and outputs.

Example 4: Python – Inventory Management System

Prompt:

Explain the following Python program that tracks inventory:

```
class Inventory:
    def __init__(self):
        self.items = {}

    def add_item(self, item_name, quantity):
        self.items[item_name] = self.items.get(item_name, 0) + quantity

    def remove_item(self, item_name, quantity):
        if item_name in self.items and self.items[item_name] >= quantity:
            self.items[item_name] -= quantity
            if self.items[item_name] == 0:
                del self.items[item_name]
        else:
            raise ValueError("Insufficient stock or item not found")

    def list_items(self):
        return self.items

inventory = Inventory()
inventory.add_item("Laptop", 5)
inventory.remove_item("Laptop", 3)
print(inventory.list_items())
```

The explanation should include:

- Purpose of the `Inventory` class.
- Explanation of each method.
- Example inputs, outputs, and edge cases (e.g., removing non-existent items).

Example 5: Java – Library Management System

Prompt:

Explain the following Java program for managing a library system:

```
import java.util.ArrayList;

public class Library {
   private ArrayList<String> books = new ArrayList<>();

   public void addBook(String book) {
      books.add(book);
   }

   public void removeBook(String book) {
      books.remove(book);
   }

   public void listBooks() {
      for (String book : books) {
         System.out.println(book);
      }
   }

   public static void main(String[] args) {
      Library library = new Library();
      library.addBook("Harry Potter");
      library.addBook("1984");
      library.listBooks();
```

```
        library.removeBook("1984");
        library.listBooks();
    }
}
```

The explanation should:

- Describe the purpose of the `Library` class.
- Explain the use of the `ArrayList`.
- Provide example scenarios where this program would be useful in industry.

Example 6: JavaScript – Data Filtering and Transformation

Prompt:

Explain the following JavaScript code that filters and transforms data:

```
const employees = [
    { id: 1, name: "Alice", department: "IT", salary: 60000 },
    { id: 2, name: "Bob", department: "HR", salary: 50000 },
    { id: 3, name: "Charlie", department: "IT", salary: 70000 }
];

const highEarners = employees
    .filter(employee => employee.salary > 55000)
    .map(employee => ({ name: employee.name, salary: employee.salary }));

console.log(highEarners);
```

The explanation should include:

- Purpose of filtering and transforming the employee dataset.

- Explanation of how `filter` and `map` are used.

- Example inputs and outputs.

Testing Code / Unit Test Cases:

Example 1: Python – Factorial Function

Prompt:

Write unit tests in Python using `unittest` for the following factorial function:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Test cases should include:

- Input: 5 (Expected output: 120)

- Input: 0 (Expected output: 1)

- Input: -1 (Expected output: Exception or error)

Example 2: Java – Palindrome Check

Prompt:

Write unit tests in Java using JUnit for the following palindrome function:

```
public boolean isPalindrome(String str) {
    String reversed = new StringBuilder(str).reverse().toString();
    return str.equals(reversed);
}
```

Test cases should include:

- Input: "radar" (Expected output: true)
- Input: "hello" (Expected output: false)
- Input: "" (Expected output: true)

Example 3: JavaScript – Maximum Element in Array

Prompt:

Write unit tests in JavaScript for the following function to find the maximum element in an array:

```
const maxElement = (arr) => Math.max(...arr);
```

Test cases should include:

- Input: [1, 3, 2] (Expected output: 3)
- Input: [] (Expected output: NaN)
- Input: [-5, -10, -2] (Expected output: -2)

Code Documentation:

Example 1: Python – Sorting Function

Prompt:

Write documentation for the following Python function:

```
def sort_array(arr):
    return sorted(arr)
```

Documentation should include:

- Description: A function to sort an array in ascending order.
- Input: A list of numbers.
- Output: A sorted list.
- Example: Input: [3, 1, 2]; Output: [1, 2, 3]

Example 2: Java – GCD Calculation

Prompt:

Write documentation for the following Java method:

```
public int gcd(int a, int b) {
```

```
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

Documentation should include:

- Description: A recursive method to calculate the GCD of two integers.
- Input: Two integers.
- Output: GCD of the two integers.
- Example: Input: 12, 8; Output: 4

Example 3: JavaScript – Filtering Even Numbers

Prompt:

Write documentation for the following JavaScript function:

```
const filterEvens = (arr) => arr.filter(num => num % 2 === 0);
```

Documentation should include:

- Description: A function to filter even numbers from an array.
- Input: An array of integers.
- Output: An array containing only even numbers.
- Example: Input: [1, 2, 3, 4]; Output: [2, 4]

Example 4: Python – User Authentication System

Prompt:

Write unit tests in Python for a user authentication system with the following functions:

```
def register_user(username, password):
```

```
    # Logic to register a user

def authenticate_user(username, password):
    # Logic to authenticate a user
```

Test cases should include:

- Successful registration and authentication of a user.
- Attempting to register a user with an existing username.
- Authenticating with an incorrect password.

Example 5: Java – E-commerce Cart

Prompt:

Write unit tests in Java for a shopping cart application with the following methods:

```
public void addItem(String item, int quantity);
public void removeItem(String item, int quantity);
public double calculateTotal();
```

Test cases should cover:

- Adding and removing items successfully.
- Calculating the total cost with multiple items.
- Handling invalid operations like removing more items than available.

Example 6: JavaScript – API Response Validator

Prompt:

Write unit tests in JavaScript to validate API responses with the following function:

```
function validateResponse(response) {
   // Logic to validate that response contains valid `status` and `data` fields
}
```

Test cases should include:

- Valid responses with `status` as `success` and `data` as non-empty.
- Invalid responses missing `status` or `data`.
- Responses with `status` as `error` and appropriate error messages.

Debugging Prompts:

Example 1: Debugging a Python Data Processing Script

Prompt:

The following Python script processes a list of transactions to calculate the total sales per product.

However, it throws a `KeyError` when executed. Debug the code step-by-step to identify and fix the issue.

Code:

```
transactions = [
   {"product": "Laptop", "quantity": 2, "price": 1500},
   {"product": "Phone", "quantity": 3, "price": 800},
   {"quantity": 1, "price": 1200}
]

sales = {}
for t in transactions:
   sales[t["product"]] += t["quantity"] * t["price"]

print(sales)
```

Instructions:

1. Analyze the structure of the input data and identify missing fields.
2. Adjust the code to handle cases where a product field is absent.
3. Re-run the script to verify the output.

Example 2: Debugging a Java Program for File Reading

Prompt:

The following Java program reads a text file and counts the number of words. However, it throws a `FileNotFoundException` even when the file exists. Debug and resolve the issue.

Code:

```
import java.io.*;
import java.util.*;
```

```
public class WordCounter {
   public static void main(String[] args) {
      File file = new File("input.txt");
      Scanner scanner = new Scanner(file);

      int wordCount = 0;
      while (scanner.hasNext()) {
         wordCount++;
         scanner.next();
      }

      System.out.println("Total words: " + wordCount);
      scanner.close();
   }
}
```

Instructions:

1. Check the file path and ensure it is correctly specified.
2. Identify if the issue lies in relative vs absolute paths.
3. Add exception handling to debug further issues if they occur.

Example 3: Debugging a JavaScript Asynchronous Function

Prompt:

The following JavaScript code fetches data from an API and processes it, but it prints `undefined` instead of the desired result. Debug and fix the issue.

Code:

```
async function fetchData() {
```

```
  let result;
  fetch("https://api.example.com/data")
    .then(response => response.json())
    .then(data => {
      result = data.name;
    });

  console.log(result);
}

fetchData();
```

Instructions:

1. Analyze the asynchronous nature of the `fetch` call.

2. Identify why `result` is not populated as expected before logging.

3. Rewrite the code using `async/await` to fix the issue and ensure correct logging of `result`.

Vibe coding examples (https://studio.firebase.google.com/):

Create an application using NextJS. In frontend user will have to provide his/her company JIRA server, his username and API token. Based on that It will give all the summary work. User has done. You have to display all the thing which user has done in bullet points.

Build a complete weather application using Next.js that shows the current weather of a city. It should:

Allow users to enter a city name and fetch the weather for that location.

Display key weather data: temperature, weather condition (e.g., sunny, cloudy), humidity, and wind speed.

Use a public weather API like OpenWeatherMap to fetch real-time data.

Style the app using Tailwind CSS or another modern CSS framework.

Handle loading states, errors (e.g., invalid city), and show a default city weather on page load (like New York).

Keep the code modular, with components like SearchBar, WeatherCard, etc.

Use Next.js API routes if needed to securely handle API keys.