Code Summarization (Java)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	<pre>Summarize the functionality of the following method: public Map < String , Integer > countWordFrequency (List < String > words) { Map < String , Integer > freqMap = new HashMap < >() ; for (String word : words) { freqMap . put (word , freqMap . getOrDefault (word , 0) + 1) ; } return freqMap ; }</pre>	The countWordFrequency method takes a list of strings (words) and returns a map (freqMap) that represents the frequency of each word in the list.	This method, countWordFrequency, takes a list of strings as input and returns a map where each unique string from the input list is a key, and its corresponding value is the number of times that string appears in the list. Essentially, it counts the frequency of each word in the provided list.
Few-shot	<pre>Example 1: def is_even(n): if n % 2 == 0: return True else: return False This method determines whether a number is even. Example 2: def is_prime(n): if n <= 1: return False for i in range(2, n): if n%i == 0: return False return True This method determines if a number is prime. Summarize the functionality of the following method: public Map < String , Integer > countWordFrequency (List < String > words) { Map < String , Integer > freqMap = new HashMap < >() ; } return false</pre>	This method counts how many times each word appears in a list.	This method calculates and returns the frequency of each unique word within a given list of words. It does this by iterating through the list and using a map to store each word as a key, with its corresponding count as the value.

```
for ( String word : words ) {
    freqMap . put ( word , freqMap . getOrDefault ( word , 0) + 1) ;
}
return freqMap;
}
```

Bug Fixing (Python – Off-by-One)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Prompt-chaining	Tell me where the off-by-one error is in this function (do not fix it): def sum_range (start , end) : total = 0 for i in range (start , end) : total += i return total -> With this code as reference: def sum_range (start , end) : total = 0 for i in range (start , end) : total = 0 for i in range (start , end) : total += i return total (Prompt for GPT) fix the off-by-one error in this line: for i in range (start , end) : (Prompt for Gemini) fix the	The off-by-one error is in this line: for i in range(start, end): This line causes the loop to exclude end, because range() in Python goes from start up to but not including end. If the intention is to include end in the sum, this creates an off-by-one error by leaving out the last value. -> To fix the off-by-one error in this line: for i in range(start, end): You should change it to: for i in range(start, end+1):	The off-by-one error is in the for loop condition: range(start, end). This will iterate from start up to, but not including, the value of end. -> def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total The range function now goes up to end + 1, ensuring that the loop includes the end value in its iteration.

	off-by-one error in the for loop condition.		
Self-consistency	Why is there an off-by-one error in this code and how can I fix it? def sum_range (start , end) : total = 0 for i in range (start , end) : total += i return total	Reason for the Off-by-One Error: The off-by-one error occurs in this line: for i in range(start, end): The range(start, end) in Python generates a sequence of numbers starting from start and up to but not including end. If you want to include the end value in the sum, it will not be included because range() excludes end from the sequence. How to Fix It: To include the end value in the loop, you should change end to end + 1 so the range will go from start to end inclusive. Here's the fix: for i in range(start, end + 1):	The off-by-one error exists because the range(start, end) function in Python generates a sequence of numbers starting from start and going up to, but not including , end. This means if you intend to sum all numbers from start to end inclusive, the loop will stop one number short. To fix this, you need to adjust the end value in the range function by adding 1. This will ensure that the sequence of numbers generated includes the intended end value. Here's the corrected code: Python def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total By changing range(start, end) to range(start, end + 1), the loop will now iterate through all the numbers from start up to and including end, thus correctly calculating the sum of the entire range.

Bug Classification (C++)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Chain-of-thought	Classify the type of bug in the	Step-by-Step Bug Classification:	Let's classify the bug in the getArray function step by step:

following C++ function. No need to provide a solution. Do it step by step: int * getArray size]; Warning: local array arr ; // Buq: returning pointer to local variable

Local Array Creation:

int arr[size];

1.

- Type of bug: Stack-based Array.
- This creates a local array arr on the stack with a size determined by the function parameter size. The array exists only within the scope of the function.

Returning Pointer to Local Array:

return arr;

2.

- Type of bug: Dangling Pointer.
- The function returns a pointer to the first element of arr, which is a local array. However, once the function scope ends, the memory for arr is deallocated as it was allocated on the stack.
- The returned pointer now points to invalid memory (a dangling pointer) because the memory for the array is no longer valid after the function returns.
- Classification: This is a bug because accessing the returned pointer after the function exits leads to undefined behavior.

Step 1: Analyze the function's intent and return type.

The function getArray aims to return a pointer (int*) to an array of integers of a given size. This suggests the expectation is to obtain a dynamically allocated array that persists beyond the function's execution.

Step 2: Examine the array declaration within the function.

The line int arr[size]; declares an integer array named arr. This declaration creates an array with automatic storage duration. This means the array is allocated on the stack, and its lifetime is limited to the scope of the getArray function.

Step 3: Analyze the return statement and its implication.

The function returns arr. In C++, when an array name is used in this context, it decays into a pointer to its first element. Thus, the function returns a pointer to the beginning of the local array arr.

Step 4: Consider the lifetime of the local variable.

Once the getArray function finishes executing, all local variables, including the array arr, are destroyed, and the memory they occupied on the stack is deallocated.

Step 5: Determine the state of the returned pointer after the function exits.

The pointer returned by getArray now points to a memory location that is no longer validly holding the array data. This pointer has become a **dangling pointer**.

Step 6: Classify the bug based on the consequences.

Returning a dangling pointer is a significant **memory management error**. Attempting to dereference this pointer in the calling code leads to **undefined behavior**, which can result in program crashes, data corruption, or other unpredictable issues.

Therefore, the type of bug in this C++ function is a **memory management error** specifically resulting in a **dangling pointer** due to returning a pointer to a local variable.

Self-consistency

What type of bug is in this C++ function? Run it on a few examples and give the most common bug experienced.

```
experienced.
int *
getArray
(int size
) {
  int arr
[ size ];
//
Warning :
local
array
  return
arr ; //
Bug:
returning
pointer
to local
variable
}
```

Type of Bug:

The **type of bug** in this C++ function is **dangling pointer**. This happens because the function returns a pointer to a local array that exists only within the function's scope. Once the function exits, the memory used by the array is deallocated, making the returned pointer point to invalid memory.

Code Analysis:

```
int * getArray (int size ) {
   int arr[size]; // Warning: local array
   return arr; // Bug: returning pointer to local
  variable
}
```

- Local Array: The array arr is created on the stack inside the function and is local to the function.
- Returning Pointer: The function returns a pointer to the first element of the array, but when the function exits, the stack frame is cleared, and the memory occupied by arr is freed.
- The returned pointer now points to invalid memory, which leads to undefined behavior if you try to dereference it.

Common Bug Experienced:

Problem:

 The most common bug encountered with this type of code is accessing invalid memory after the function call. Since the array is local to the function and is The type of bug in this C++ function is a **memory management error**, specifically resulting in a **dangling pointer**.

Let's consider what happens when this function is used:

Example 1:

C++

```
#include <iostream>
```

```
int * getArray (int size ) {
  int arr [ size ];
  std::cout << "Inside getArray: Address of arr = " << static_cast<void*>(arr) << std::endl;
  for (int i = 0; i < size; ++i) {
    arr[i] = i * 10;
  }
  return arr;
}

int main() {
  int size = 3;
  int *myArray = getArray(size);
  std::cout << "In main: Address received = " << static_cast<void*>(myArray) << std::endl;
  std::cout << "Attempting to access array elements in main:" << std::endl;
  for (int i = 0; i < size; ++i) {
    std::cout << myArray[i] << " ";
  }
  std::cout << std::endl;
  return 0;
}</pre>
```

Likely Output (behavior is undefined):

Inside getArray: Address of arr = 0x7ffeefc7c8d0 In main: Address received = 0x7ffeefc7c8d0 Attempting to access array elements in main: 0 10 20 deallocated when the function exits, any attempt to use the returned pointer results in accessing memory that is no longer valid. This often leads to segmentation faults, corrupted data, or undefined behavior.

Example Runs and Output:

Let's consider running this code in various examples.

Example 1:

```
int* array = getArray(5);
std::cout << array[0] << std::endl; // Undefined
behavior (dangling pointer)</pre>
```

1. Expected Outcome:

 The program might crash with a segmentation fault or output garbage values due to accessing memory that has been freed.

Example 2:

```
int* array = getArray(10);
for (int i = 0; i < 10; i++) {
   std::cout << array[i] << " "; // Undefined behavior
(dangling pointer)
}</pre>
```

2. Expected Outcome:

 The loop may print random values, or it could crash due to accessing invalid memory. In this scenario, it might appear to work correctly because the memory on the stack where arr was located hasn't been immediately overwritten. However, this is not guaranteed.

Example 2:

```
C++
```

```
#include <iostream>
```

```
int * getArray (int size ) {
int arr [ size ];
 std::cout << "Inside getArray (first call): Address of arr = " << static cast<void*>(arr) << std::endl;
 for (int i = 0; i < size; ++i) {
  arr[i] = i * 10;
return arr;
void anotherFunction() {
int anotherArr[5] = \{1, 2, 3, 4, 5\};
std::cout << "Inside anotherFunction: Address of anotherArr = " << static cast<void*>(anotherArr)
<< std::endl:
int main() {
int size = 3:
 int *myArray = getArray(size);
 anotherFunction(); // Call another function that might overwrite the stack
 std::cout << "In main: Address received = " << static cast<void*>(myArray) << std::endl;
 std::cout << "Attempting to access array elements in main (after anotherFunction):" << std::endl:
 for (int i = 0; i < size; ++i) {
  std::cout << myArray[i] << " ";
 std::cout << std::endl:
return 0:
```

More Likely Output (behavior is undefined):

Inside getArray (first call): Address of arr = 0x7ffee3b1c8d0 Inside anotherFunction: Address of anotherArr = 0x7ffee3b1c8a0

Why It Happens:

- Stack Allocation: The array arr is allocated on the stack, and once the function ends, the stack is unwound, and the memory used by arr is reclaimed.
- Returning Local Pointer: Returning a pointer to a local array (or any local variable) is dangerous because it points to a location that will no longer hold valid data once the function finishes execution.

Conclusion:

The bug is that the function **returns a pointer to a local array**, leading to **undefined behavior** when accessing the array after the function exits, commonly resulting in segmentation faults or corrupted data.

In main: Address received = 0x7ffee3b1c8d0
Attempting to access array elements in main (after anotherFunction): 1 2 3

In this case, after getArray returns, anotherFunction is called. This function allocates its own local array anotherArr on the stack. It's highly likely that this allocation will overwrite the memory that was previously occupied by arr in getArray. Consequently, when main tries to access the elements pointed to by myArray, it reads whatever data happens to be in that memory location at that time, which is likely the data from anotherArr or some other unrelated data.

Most Common Bug Experienced:

The most common bug experienced when using this function is **reading garbage values** or **incorrect data** when trying to access the array elements through the returned pointer in the calling function. This is because the memory the pointer points to is no longer guaranteed to hold the original data. In more severe cases, it can lead to **program crashes** due to accessing memory that is no longer considered valid for the program to read.

While the first example might coincidentally "work," it's crucial to understand that this is undefined behavior and highly unreliable. The second example demonstrates the more typical and problematic outcome.

Generating Email Validators (Python + Regex)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	Complete the function using regex to validate basic email addresses: def is_valid_email (email):	import re def is_valid_email(email): pattern = r'^[a-zA-Z0-9%+-]+@[a-zA-Z0-9]+\ .[a-zA-Z]{2,}\$' return re.fullmatch(pattern, email) is not None	import re def is_valid_email(email): """Basic email validation using a simple regex.""" return bool(re.match(r"[^@]+@[^@]+\.[^@]+", email))

```
Self-consisten
                                                                                 Python
                 Complete the
                 function in several
CV
                 ways using regex to
                                         Option 1: Simple regex
                                                                                 import re
                 validate basic email
                                        import re
                 addresses, and
                                                                                 def is valid email regex 1(email):
                 return the most
                                                                                  """Basic email validation using a simple regex."""
                                        def is_valid_email(email):
                 efficient:
                                                                                  return bool(re.match(r"[^@]+@[^@]+\.[^@]+", email))
                                           return
                                        re.fullmatch(r"[^@]+@[^@]+\.[^@]+",
                 is valid email
                                                                                 def is valid email regex 2(email):
                                        email) is not None
                                                                                   """Slightly more robust basic email validation."""
                                                                                  return bool(re.match(r"^[a-zA-Z0-9. %+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$", email))
                                        Pros: Fast and very basic Cons: Accepts some invalid
                                                                                 def is valid email regex 3(email):
                                                                                  """Another common basic email validation regex."""
                                        formats like a@b.c@d
                                                                                  return bool(re.match(r"^[^.\s][\w.-]+(\.[\w-]+)+", email))
                                                                                 def is_valid_email_regex_4(email):
                                                                                  """Basic validation allowing for more characters."""

    Option 2: Stricter basic regex

                                                                                  return bool(re.match(r"^[\s@]+@[^\s@]+\.[^\s@]+$", email))
                                        import re
                                                                                 def is valid email(email):
                                                                                  """Placeholder function using the most efficient regex."""
                                        def is valid email(email):
                                                                                  return is valid email regex 1(email) # Initially set to the simplest
                                        re.fullmatch(r"[a-zA-Z0-9. %+-]+@[a-
                                                                                 # --- Efficiency Comparison (Crude) ---
                                        zA-Z0-9.-]+\.[a-zA-Z]{2,}", email) is
                                                                                 import time
                                        not None
                                                                                 def test efficiency(func, emails, iterations=100000):
                                                                                  start time = time.time()
                                        Pros: Covers common valid
                                                                                  for in range(iterations):
                                        emails
                                                                                    for email in emails:
                                         X Cons: Still not RFC-compliant,
                                                                                     func(email)
                                        but sufficient for most practical needs
                                                                                  end time = time.time()
                                                                                  return end time - start time
                                                                                 valid emails = ["test@example.com", "user.name@sub.domain.co.uk", "a+b@example-hyphen.com"]
                                                                                 invalid emails = ["invalid", "no-at-sign.com", "missing.dot@", "@missing-local", "local@.missing-domain"]

    Option 3: Very strict basic regex
```

```
(avoids consecutive dots,
                                         all emails = valid emails + invalid emails
leading/trailing hyphens)
                                         print(f"Efficiency Test (lower is better, {len(all emails) * 100000} evaluations):")
import re
                                         regex functions = [
                                            is valid email regex 1,
def is valid email(email):
                                            is valid email regex 2,
  pattern =
                                            is valid email regex 3,
r'''(?!.*\.\.)[a-zA-Z0-9. %+-]+@(?:(?!-)
                                            is valid email regex 4,
[a-zA-Z0-9-]{1,63}(?<!-)\.)+[a-zA-Z]{2,}
                                         for func in regex functions:
  return re.fullmatch(pattern, email) is
                                           duration = test_efficiency(func, all_emails)
not None
                                           print(f"\{func. name \}: \{duration:.4f\} seconds"\)
                                         print("\nNote on Efficiency:")
Pros: Prevents common
                                         print("The efficiency of regex can depend on the complexity of the pattern and the regex engine.")
edge-case issues like double dots or
                                         print("For *basic* email validation, simpler regex patterns are generally more efficient.")
leading hyphens
                                         print("'is valid email regex 1' is likely the most efficient among these as it performs the fewest checks.")
X Cons: More complex, slightly
                                         print("However, it's also the least strict in terms of email format accuracy.")
slower
                                         print("The 'most efficient' choice depends on the level of validation rigor required.")
                                         print("For truly robust email validation, consider more comprehensive (and potentially less efficient) regex
                                         patterns or dedicated email validation libraries.")
Most Efficient (Recommended):
                                         # Update the main 'is valid email' function to use the most efficient (simplest) regex
Option 2
                                         def is valid email(email):
                                           """Basic email validation using the most efficient (simplest) regex."""
import re
                                           return is valid email regex 1(email)
def is valid email(email):
  return
                                         The is valid email function is initially set to use is valid email regex 1 as it is likely the most efficient of the
re.fullmatch(r"[a-zA-Z0-9. %+-]+@[a-
zA-Z0-9.-]+\[a-zA-Z]\{2,\}, email) is
                                         basic patterns presented. You can change this to one of the other regex functions if you require a slightly
not None
                                         stricter level of validation at a potential cost to performance.
```

Generating Flask APIs (Python)

Prompt method: Prompt: GPT-4 output: Gemini output:

Zero-shot	Create a '/greet/' endpoint that returns a JSON greeting: from flask import Flask , jsonify app = Flask (name) @app . route ('/ greet / < username >') def greet (username) : # TODO : Return a JSON greeting pass	from flask import Flask, jsonify app = Flask(name) @app.route('/greet/ <u sername="">') def greet(username): return jsonify({'message': f'Hello, {username}!'}) # Optional: if you want to run the app directly ifname == 'main': app.run(debug=True)</u>	from flask import Flask, jsonify app = Flask(name) @app.route('/greet/ <username>') def greet(username): """Returns a JSON greeting with the provided username.""" greeting_message = f'Hello, {username}! Welcome to Williamsburg, Virginia!" return jsonify({'greeting': greeting_message, 'location': 'Williamsburg, Virginia, United States'}) ifname == 'main': app.run(debug=True)</username>
Chain-of-thought	Create a '/greet/' endpoint that returns a JSON greeting. Do it step by step: from flask import Flask , jsonify app = Flask (name) @app . route ('/ greet / < username >') def greet (username) : # TODO : Return a JSON	✓ Step 1: Import necessary modules You've already done this part correctly: from flask import Flask, jsonify ✓ Step 2: Create a Flask app instance This initializes your Flask application:	Step 1: Import necessary modules. We already have this in your provided code: Python from flask import Flask, jsonify This line imports the Flask class for creating the application and the jsonify function for converting data to JSON. Step 2: Create a Flask application instance. Again, this is already done: Python app = Flask(name)

greeting pass app =
Flask(__name__)

Step 3: Define the route with a dynamic URL parameter

Make sure to use standard quotes (' or ") instead of curly quotes ('), which can cause errors.

@app.route('/greet/<u sername>')

This route captures the value after /greet/ and assigns it to the username variable.

Step 4: Define the view function

You already started it. Now let's complete it by returning a JSON response:

def greet(username):
 return
jsonify({'message':
f'Hello, {username}!'})

This line initializes our Flask web application.

Step 3: Define the route and associated function.

You've also set up the route:

Python

```
@app.route('/greet/<username>')
def greet(username):
  # TODO : Return a JSON greeting
pass
```

The @app.route('/greet/<username>') decorator tells Flask that when a user accesses a URL that matches /greet/ followed by some text (which Flask will treat as the username), the greet function should be executed.

Step 4: Implement the greet function to return a JSON response.

Inside the greet function, we need to create a Python dictionary containing the greeting and then convert it to a JSON response using jsonify(). Since we know the current location is Williamsburg, Virginia, United States, let's include that in the greeting.

Python

```
@app.route('/greet/<username>')
def greet(username):
   greeting_message = f"Hello, {username}! Welcome to Williamsburg, Virginia, United States!"
   response_data = {'greeting': greeting_message, 'location': 'Williamsburg, Virginia, United States'}
   return jsonify(response_data)
```

In this step:

- We create a formatted string greeting message that includes the username captured from the URL.
- We create a Python dictionary response_data with two keys: 'greeting' and 'location', holding the greeting
 message and the current location, respectively.

```
We use jsonify(response_data) to convert this dictionary into a JSON response that Flask will send back to the
                                 client.
isonify creates a
proper JSON
response, including
                        Step 5: Add the code to run the Flask development server (optional but necessary for testing).
setting the correct
content-type header.
                        To run the application, you'll typically add the following block at the end of your script:
                        Python
V Final Full
Working Code:
                        if name == ' main ':
                         app.run(debug=True)
from flask import
Flask, jsonify
app =
                        This ensures that the Flask development server starts only when the script is executed directly. The debug=True option
Flask(__name___)
                        enables debugging features.
@app.route('/greet/<u
                        Complete Code:
sername>')
def greet(username):
                        Putting it all together, your app.py file should look like this:
  return
isonify({'message':
f'Hello, {username}!'})
                        Python
# Optional: run the
                        from flask import Flask, isonify
app
if name ==
' main ':
                        app = Flask( name )
                        @app.route('/greet/<username>')
app.run(debug=True)
                        def greet(username):
                         greeting message = f"Hello, {username}! Welcome to Williamsburg, Virginia, United States!"
                         response data = {'greeting': greeting message, 'location': 'Williamsburg, Virginia, United States'}
                         return jsonify(response data)
                        if __name__ == '__main__':
                         app.run(debug=True)
```

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Prompt method: Prompt-chaining	Using the outline, write the schema for a review app with users in psuedocode: TODO: Design schema with appropriate keys and constraints Tables: users (id, name), books (id, title), reviews (id, user_id, book_id, rating) -> Using this psuedocode, implement the schema for a review app with users, books, and reviews: [pseudocode provided by previous response]	USERS TABLE CREATE TABLE users (id INTEGER PRIMARY KEY, Unique user ID name TEXT NOT NULL User's name); BOOKS TABLE CREATE TABLE books (id INTEGER PRIMARY KEY, Unique book ID title TEXT NOT NULL Book title); REVIEWS TABLE CREATE TABLE reviews (id INTEGER PRIMARY KEY, Unique review ID user_id INTEGER NOT NULL, FK: references users(id) book_id INTEGER NOT NULL, FK: references books(id) rating INTEGER NOT NULL, Foreign key constraints FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (book_id) REFERENCES books(id), Optional constraint: a user can only review a book once	Schema for a Review App Table: users CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT, Unique identifier for each user, automatically generated name VARCHAR(255) NOT NULL, User's name, cannot be empty Add other user-related fields as needed (e.g., email, registration_date) email VARCHAR(255) UNIQUE, Optional: User's email, must be unique if present registration_date DATETIME DEFAULT CURRENT_TIMESTAMP Optional: Date when the user registered); Table: books CREATE TABLE books (id INTEGER PRIMARY KEY AUTOINCREMENT, Unique identifier for each book, automatically generated title VARCHAR(255) NOT NULL UNIQUE, Title of the book, cannot be empty and should be unique Add other book-related fields as needed (e.g., author, publication_year, ISBN) author VARCHAR(255), Optional: Author of the book isbn VARCHAR(20) UNIQUE Optional: Year the book was published isbn VARCHAR(20) UNIQUE Table: reviews CREATE TABLE reviews (id INTEGER PRIMARY KEY AUTOINCREMENT, Unique identifier for each review, automatically generated user_id INTEGER NOT NULL, Foreign key referencing the users table book_id INTEGER NOT NULL, Foreign key referencing the users table Constraints for data integrity CREIGN KEY (user_id) REFERENCES users(id), Ensures that the user_id exists in the users table FOREIGN KEY (book_id) REFERENCES books(id), Ensures that the rating is within a valid range (e.g., 1 to 5)
	previous		FOREIGN KEY (book_id) REFERENCES books(id), Ensures that the book_id exists in the books table

```
CREATE TABLE users (
                                    CREATE INDEX idx rating ON reviews (rating);
 id INTEGER PRIMARY KEY
AUTOINCREMENT. --
                                    ->
Auto-incrementing user ID
 name TEXT NOT NULL
                                    -- Schema for a Review App
-- User's name
                                    -- Table: users
                                    CREATE TABLE users (
-- BOOKS TABLE
                                     id INTEGER PRIMARY KEY AUTO INCREMENT,
CREATE TABLE books (
                                     name VARCHAR(255) NOT NULL,
 id INTEGER PRIMARY KEY
                                     email VARCHAR(255) UNIQUE,
AUTOINCREMENT. --
                                     registration date DATETIME DEFAULT CURRENT TIMESTAMP
Auto-incrementing book ID
 title TEXT NOT NULL
Book title
                                    -- Table: books
                                    CREATE TABLE books (
                                     id INTEGER PRIMARY KEY AUTO INCREMENT,
-- REVIEWS TABLE
                                     title VARCHAR(255) NOT NULL UNIQUE.
CREATE TABLE reviews (
                                     author VARCHAR(255),
 id INTEGER PRIMARY KEY
                                     publication year INTEGER,
                                     isbn VARCHAR(20) UNIQUE
AUTOINCREMENT, --
Auto-incrementing review ID
 user id INTEGER NOT NULL,
-- References users(id)
                                    -- Table: reviews
 book id INTEGER NOT NULL,
                                    CREATE TABLE reviews (
-- References books(id)
                                     id INTEGER PRIMARY KEY AUTO INCREMENT,
 rating INTEGER NOT NULL
                                     user id INTEGER NOT NULL,
CHECK(rating BETWEEN 1 AND 5). --
                                     book id INTEGER NOT NULL.
Rating 1-5
                                     rating INTEGER NOT NULL,
                                     review text TEXT,
                                     created at DATETIME DEFAULT CURRENT TIMESTAMP,
  -- Ensure the same user can't review
the same book more than once
                                     FOREIGN KEY (user id) REFERENCES users(id),
                                     FOREIGN KEY (book id) REFERENCES books(id),
 UNIQUE (user id, book id),
                                     CONSTRAINT rating range CHECK (rating >= 1 AND rating <= 5).
  -- Foreign key constraints
                                     CONSTRAINT unique user book UNIQUE (user id, book id)
  FOREIGN KEY (user id)
REFERENCES users(id),
 FOREIGN KEY (book id)
                                    -- Optional Indexes for performance (improve guery speed)
REFERENCES books(id)
                                    CREATE INDEX idx user id ON reviews (user id);
                                    CREATE INDEX idx book id ON reviews (book id);
);
                                    CREATE INDEX idx rating ON reviews (rating);
```

7	\\\/::t = 41= =	LICEDO TADI E	Och and for a Davison Ann
Zero-shot	Write the schema for a	USERS TABLE CREATE TABLE users (Schema for a Review App
		id INTEGER PRIMARY KEY	Table: users
	review app with users, books,	AUTOINCREMENT, Unique ID for	CREATE TABLE users (
	and reviews:	each user	id INTEGER PRIMARY KEY AUTO_INCREMENT, Unique identifier for each user
	TODO :	name TEXT NOT NULL	name VARCHAR(255) NOT NULL, User's name
	Design schema	User's name	Add other user-related fields as needed (e.g., email, registration date)
	with		email VARCHAR(255) UNIQUE, Optional: User's email, must be unique if present
	appropriate);	registration date DATETIME DEFAULT CURRENT TIMESTAMP Optional: Date when the user
	keys and	BOOKS TABLE	registered
	constraints	CREATE TABLE books ();
	Tables :	id INTEGER PRIMARY KEY	<i>"</i>
	users (id ,	AUTOINCREMENT, Unique ID for	Table: books
	name), books	each book	CREATE TABLE books (
	(id , title),	title TEXT NOT NULL	id INTEGER PRIMARY KEY AUTO INCREMENT, Unique identifier for each book
	reviews (id ,	Book title	title VARCHAR(255) NOT NULL UNIQUE, Title of the book, must be unique
	user id,);	Add other book-related fields as needed (e.g., author, publication year, ISBN)
	book_id , rating	,	author VARCHAR(255), Optional: Author of the book
)	REVIEWS TABLE	publication_year INTEGER, Optional: Year the book was published
	'	CREATE TABLE reviews (isbn VARCHAR(20) UNIQUE Optional: ISBN of the book, must be unique if present
		id INTEGER PRIMARY`KEY);
		AUTOINCREMENT, Unique ID for	
		each review	Table: reviews
		user_id INTEGER NOT NULL,	CREATE TABLE reviews (
		Foreign key to users.id	id INTEGER PRIMARY KEY AUTO_INCREMENT, Unique identifier for each review
		book_id INTEGER NOT NULL,	user_id INTEGER NOT NULL, Foreign key referencing the users table
		Foreign key to books.id	book_id INTEGER NOT NULL, Foreign key referencing the books table
		rating INTEGER NOT NULL CHECK	rating INTEGER NOT NULL, Rating given by the user for the book
		(rating BETWEEN 1 AND 5), Rating:	review_text TEXT, Optional: Text content of the review
		1 to 5	created_at DATETIME DEFAULT CURRENT_TIMESTAMP, Date and time when the review was created
			Constraints for data integrity
		Constraints	FOREIGN KEY (user_id) REFERENCES users(id), Ensures user exists
		FOREIGN KEY (user_id)	FOREIGN KEY (book_id) REFERENCES books(id), Ensures book exists
		REFERENCES users(id),	CONSTRAINT rating_range CHECK (rating >= 1 AND rating <= 5), Ensures rating is within valid range
		FOREIGN KEY (book_id)	CONSTRAINT unique_user_book UNIQUE (user_id, book_id) Ensures a user can only review a book
		REFERENCES books(id),	once
		UNIQUE (user_id, book_id));
		Prevents duplicate reviews per	
		user/book	Optional Indexes for performance (improve query speed)
]);	CREATE INDEX idx_user_id ON reviews (user_id);
			CREATE INDEX idx_book_id ON reviews (book_id);
			CREATE INDEX idx_rating ON reviews (rating);

Null Dereference Detection (Java)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Chain-of-thought	<pre>Identify any null dereference risk. Do it step by step: public int getLength (String s) { return s . length (); // What if s is null ? }</pre>	✓ Step 1: Understand what the method does This method attempts to return the length of the string s using s.length(). ✓ Step 2: Consider the input • The parameter String s is not guaranteed to be non-null unless explicitly checked or annotated (e.g., with @NonNull). • Java allows method parameters to be null unless explicitly prevented.	Step 1: Understand the function's purpose. The function getLength takes a String object's as input and is intended to return the length of that string. Step 2: Examine the operation performed on the input. The core operation is s.length(). This method is called on the String object's to retrieve its length. Step 3: Consider the possible values of the input parameter's. The input parameter's is declared as a String. In Java (and many other object-oriented languages), a reference variable of an object type (like String) can hold one of two types of values: • A reference to an actual String object in memory. • The special value null, indicating that the variable does not currently refer to any object.
		 ✓ Step 3: What happens if s is null? If s == null, then s.length() attempts to dereference a null reference. This leads to a NullPointerException (NPE) at runtime. 	Step 4: Evaluate the behavior of the operation when s is null. If the s parameter is null, then attempting to call the length() method on it (s.length()) will result in a NullPointerException. This is because you are trying to access a member (the length() method) of something that doesn't exist (no object is being referenced). Step 5: Identify the null dereference risk. Based on the above analysis, there is a null dereference risk in the getLength function. If the String object passed as the argument s is null, the line return s.length(); will throw a NullPointerException.

		Yes, this method contains a null dereference risk. The risk comes from this line: return s.length(); It directly calls .length() on s without checking whether s is null.	Conclusion: The getLength function has a null dereference risk because it directly calls the length() method on the input String object s without checking if s is null. If s is null, this will lead to a NullPointerException.
Self-consistency	Use several examples to identify dereference instances in this code, and identify the risk: public int getLength (String s) {	✓ Step-by-step Examples: Example 1: Valid String getLength("hello") • s = "hello"	Scenario 1: Passing a valid String object Java String text = "Hello"; int length = getLength(text); System.out.println("Length: " + length); // Output: Length: 5 In this case, the s parameter inside the getLength function refers to a valid String object
	<pre>return s . length () ; // What if s is null ? }</pre>	 s.length() → 5 No error, returns 5. Example 2: Empty String getLength("")	("Hello"). The s.length() call successfully retrieves the length of the string, and the function returns 5. There is no dereference error here. Scenario 2: Passing a null String reference Java String text = null; int length = getLength(text); // This line will throw a NullPointerException System.out.println("Length: " + length); // This line will not be reached
		• $s = ""$ • $s.length() \rightarrow 0$	Here, the text variable is explicitly set to null. When getLength(text) is called, the s parameter inside the function is null. The line return s.length(); attempts to call the length() method on a

• **No error**, returns 0.

Example 3: Null String getLength(null)

- s = null
- s.length() → X Throws
 NullPointerException
- This is the dereference risk accessing .length() on a null reference.

Risk Summary:

- Dereference Point: s.length()
- When Risk Happens: When s == null
- Exception Thrown: NullPointerException
- Why It's Risky: It can crash your program if not caught.

Conclusion:

null reference. This is an invalid operation and results in a **NullPointerException**. The program will terminate (unless the exception is caught and handled).

Scenario 3: String obtained from an operation that might return null

Java

String name = System.getProperty("user.name"); // May return null if the property is not set int nameLength = getLength(name); // Potential NullPointerException here System.out.println("Name length: " + nameLength);

In this example, System.getProperty("user.name") might return a String object containing the user's name, but it could also return null if the system property is not defined. If it returns null, the subsequent call to getLength(name) will lead to a **NullPointerException**.

Dereference Instance:

The dereference instance in the getLength function occurs at the line:

Java

return s.length();

Here, the code attempts to access a member (length()) of the object referenced by the variable s.

Risk:

The primary risk associated with this dereference instance is a **NullPointerException**. This is a runtime exception that occurs when you try to access a member (field or method) of an object through a null reference.

Consequences of NullPointerException:

Program Termination: If a NullPointerException is not caught and handled, it will cause the program to abruptly terminate.

The line return s.length(); dereferences s and assumes it is not null. This poses a null dereference risk, which occurs if the method is called with a null argument.

Let me know if you want to see safe ways to handle this, like default values or exception handling!

- Loss of Data: If the program was in the middle of an important operation, data might be lost.
- Poor User Experience: For end-users, the application will crash, leading to a frustrating experience.
- Debugging Challenges: While the location of a NullPointerException is usually clear from the stack trace, identifying the exact reason why a variable became null can sometimes require careful debugging.

In summary, the <code>getLength</code> function, as written, is susceptible to a <code>NullPointerException</code> if a <code>null String</code> reference is passed as an argument. This highlights the importance of null checks when dealing with object references in programming.

CSV Parser Variants (Python)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
	Improve the parser to support quoted fields: def parse_csv_line (line) : return line . split (',') # Incomplete : doesn 't handle quoted fields	To properly handle quoted fields in a CSV line—such as: John,Doe,"New York, NY",35 —you'll need to enhance the parser so that it respects commas inside quotes. Here's an improved version using Python's csv module (which is built to handle this correctly): Recommended Approach: Use	def parse_csv_line_improved(line): Parses a CSV line, correctly handling quoted fields. Args: line: A string representing a single line from a CSV file. Returns: A list of strings, where each string is a field from the CSV line. """ # Regex to find fields, handling quoted values # Explanation: # (?: # Non-capturing group # "([^n]*)" # Match a quoted field (content inside double quotes) - Group 1 # # OR # ([^n]*) # Match a non-quoted field (any character except comma) - Group 2 #) # End of non-capturing group # (?:, \$) # Match a comma OR the end of the line (non-capturing group)

pattern = r'(?: "([^"]*)" | ([^,]*)) (?:,|\$) ' csv.reader fields = re.findall(pattern, line) import csv # Process the result to get the actual field values from io import StringIO return [field[0] if field[0] else field[1] for field in fields] def parse csv line(line): reader = csv.reader(StringIO(line)) return next(reader) Chain-of-thought Improve the parser to support quoted fields. Do it Step 1: Understand the step-by-step: Problem parse csv line Input: A CSV line like 'John, Doe, "New York, NY",35' split (',') # Naive split(',') gives: ['John', 'Doe', '"New York', ' NY"', '35'1 X Python Goal: Return: ['John', fields = [] 'Doe', 'New current field = "" York, NY', in quotes = False '35'] 🔽 for char in line:

Ntep 2: Initialize

Step 1: Understand the Problem

The basic line.split(',') method simply splits the string at every comma. This works for simple CSV files where fields don't contain commas. However, if a field itself needs to contain a comma, it's typically enclosed in double quotes. Our improved parser needs to recognize these quoted fields and treat the commas within them as part of the field value, not as delimiters.

Step 2: Consider a State-Based Approach

One way to handle this is to iterate through the CSV line character by character and keep track of whether we are currently inside a quoted field or not.

Step 3: Implement the State Machine

```
def parse csv line improved v1(line):
  if char == "":
   in guotes = not in guotes # Toggle the in guotes state
  elif char == ',' and not in quotes:
   fields.append(current field)
   current field = ""
  else:
```

Variables

We'll track:

- The current field
- Whether we're inside quotes
- A list to hold the parsed fields

Here's the improved parser:

```
def parse csv line(line):
  fields = []
  field = "
  in quotes = False
  i = 0
  while i < len(line):
    char = line[i]
    if char == "":
       if in quotes and i + 1
< len(line) and line[i + 1] ==
          # Escaped quote:
"" inside quoted field
          field += ""
          i += 1
       else:
          in quotes = not
in quotes # Toggle quote
mode
     elif char == ',' and not
```

```
current_field += char
fields.append(current_field) # Add the last field
return fields
```

In this version:

- We initialize an empty list fields to store the parsed fields, an empty string current_field to build the current field's value, and a boolean in quotes to track if we are inside a quoted section.
- We iterate through each character of the input line.
- If we encounter a double quote ("), we toggle the in quotes state.
- If we encounter a comma (,) and we are *not* currently inside quotes (not in_quotes), it means the comma is a delimiter, so we append the current field to our fields list and reset current field.
- For any other character (including commas inside quotes), we append it to the current field.
- After processing the entire line, we append the final current_field to the fields list, as there won't be a trailing comma for the last field.

Step 4: Handle Potential Edge Cases (Optional but Recommended)

The above version handles basic quoted fields. However, real-world CSV can have more complexities, such as:

- Escaped Quotes: Sometimes, double quotes within a quoted field are escaped (e.g., "").
- Leading/Trailing Whitespace: Fields might have leading or trailing whitespace.

Let's improve the parser to handle escaped quotes ("" becomes "):

Python

```
def parse_csv_line_improved_v2(line):
    fields = []
    current_field = ""
    in_quotes = False
    i = 0
    while i < len(line):
        char = line[i]
    if char == "":
        if i + 1 < len(line) and line[i + 1] == "":
        current_field += "" # Treat "" as a single " inside quotes
        i += 1 # Skip the next quote
    else:</pre>
```

```
in_quotes:
            fields.append(field)
            field = "
            else:
            field += char
            i += 1
```

fields.append(field) # Add the last field return fields

Step 4: Test It

print(parse_csv_line('John,D oe,"New York, NY",35'))
➤ ['John', 'Doe', 'New York, NY', '35']

```
in_quotes = not in_quotes
elif char == ',' and not in_quotes:
  fields.append(current_field.strip()) # Strip whitespace
  current_field = ""
else:
  current_field += char
  i += 1
fields.append(current_field.strip()) # Add and strip the last field
return fields
```

In this improved version:

- We use a while loop with an index i to allow us to look ahead.
- When we encounter a double quote:
 - We check if the next character is also a double quote. If it is, we treat it as an escaped quote
 within the field and append a single double quote to current_field, advancing the index i by one to
 skip the next quote.
 - Otherwise, it's a regular quote toggling the in_quotes state.
- When we encounter a comma outside of quotes, we append the current_field to fields after stripping any leading or trailing whitespace using .strip().
- We also strip whitespace from the final current field before appending it.

Step 5: Consider Using the csv Module (Python Standard Library - Recommended for Robustness)

Python's standard library provides a csv module that is specifically designed for parsing CSV files and handles various complexities and dialects. This is generally the most robust and recommended approach for real-world CSV parsing.

Python

import csv

```
def parse_csv_line_using_csv(line):
  reader = csv.reader([line]) # Create a reader for a single line
  for row in reader:
    return list(row) # Return the row as a list
    return [] # Handle empty lines
```

Data Class to API Conversion (Kotlin)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	Convert the data class to a REST API using Ktor: data class Product (val id : Int , val name : String , val price : Double) // TODO : Create GET and POST endpoints using Ktor	import io.ktor.application.* import io.ktor.features.* import io.ktor.http.* import io.ktor.request.* import io.ktor.response.* import io.ktor.response.* import io.ktor.serialization.* import kotlinx.serialization.* import kotlinx.serialization.son.* @kotlinx.serialization.Serializable data class Product(val id: Int, val name: String, val price: Double) val products = mutableListOf <product>() fun Application.module() { install(ContentNegotiation) { json() } routing { route("/products") { // GET /products get { call.respond(products) } // POST /products post { val product = call.receive<product>() products.add(product) call.respond(HttpStatusCode.Created, product) }</product></product>	<pre>import io.ktor.server.application.* import io.ktor.server.response.* import io.ktor.server.netty.* import io.ktor.server.netty.* import io.ktor.server.netty.* import io.ktor.server.plugins.contentnegotiation.* import io.ktor.server.request.* import kotlinx.serialization.Serializable import io.ktor.http.HttpStatusCode @Serializable data class Product(val id: Int, val name: String, val price: Double) // In-memory storage for products (for demonstration purposes) val products = mutableListOf<product>(Product(1, "Laptop", 1200.0), Product(2, "Mouse", 25.0), Product(3, "Keyboard", 75.0)) fun main(args: Array<string>) { io.ktor.server.netty.EngineMain.main(args) } fun Application.module() { // Install the ContentNegotiation plugin to handle JSON serialization</string></product></pre>

```
install(ContentNegotiation) {
          call.respond(products)
          val id = call.parameters["id"]?.toIntOrNull()
              call.respond(HttpStatusCode.BadRequest, "Product ID must be
an integer.")
              call.respond(product)
              call.respond(HttpStatusCode.NotFound, "Product not found.")
      post("/products") {
```

```
call.receive<Product>()
data format. Expected JSON: { id: Int, name: String, price: Double }")
           if(product.id <= 0 || product.name.isBlank() || product.price</pre>
<= 0) {
data. id and price must be positive, name cannot be blank.")
already exists.")
           products.add(product)
           call.respond(HttpStatusCode.Created, product) // Respond with
```

Chain-of-thought

Convert the data class to a REST API using Ktor. Do it step by step:

data class Product (val id : Int , val name : String , val price : Double) Create GET and POST endpoints using Ktor

Step 1: Set Up Dependencies

Make sure you have Ktor and necessary dependencies in your build.gradle.kts or pom.xml. Here's what you'll need in build.gradle.kts:

dependencies {

implementation("io.ktor:ktor-server-core:2. 2.4")

implementation("io.ktor:ktor-server-netty:2. 2.4")

implementation("io.ktor:ktor-serialization:2. 2.4")

implementation("io.ktor:ktor-server-content -negotiation:2.2.4")

implementation("org.jetbrains.kotlinx:kotlin x-serialization-json:1.3.2")

testImplementation("io.ktor:ktor-server-test s:2.2.4")

Step 2: Create the Product Data Class

Here is the Product data class:

import kotlinx.serialization.Serializable

@Serializable data class Product(val id: Int. val name: String, val price: Double)

```
import io.ktor.server.application.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import io.ktor.server.netty.* // Use Netty
import io.ktor.serialization.kotlinx.json.* // Use kotlinx-json
import io.ktor.server.plugins.contentnegotiation.* // For
ContentNegotiation
import io.ktor.server.request.* // For receiving request bodies
import kotlinx.serialization.Serializable // For JSON serialization
import io.ktor.http.HttpStatusCode // For HTTP status codes
// 1. Define the Data Class and Mark it Serializable
@Serializable
data class Product(val id: Int, val name: String, val price: Double)
// 2. In-Memory Data Storage (Replace with a database in a real app)
val products = mutableListOf<Product>(
   Product(1, "Laptop", 1200.0),
   Product(2, "Mouse", 25.0),
   Product(3, "Keyboard", 75.0)
```

 We add @Serializable to the class to allow it to be serialized and deserialized to/from JSON.

♦ Step 3: Create a Ktor Application

Now, let's set up the basic Ktor application with the endpoints.

- Define the Product list to store products (for simplicity, we'll keep it in memory).
- 2. **Define the routes** to handle **GET** and **POST** requests.

```
import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.http.HttpStatusCode
import io.ktor.request.receive
import io.ktor.response.respond
import io.ktor.routing.*
import
io.ktor.server.engine.embeddedServer
import io.ktor.server.netty.Netty
import
io.ktor.server.plugins.contentnegotiation.C
ontentNegotiation
import
io.ktor.server.plugins.statuspages.StatusP
ages
import io.ktor.server.testing.testApplication
import io.ktor.server.server.json
```

val products = mutableListOf<Product>()
// In-memory list to store products

```
// 3. Application Entry Point (main function)
fun main(args: Array<String>) {
  io.ktor.server.netty.EngineMain.main(args) // Start Netty
// 4. Application Module (Configure Ktor)
fun Application.module() {
  // 4.1 Install Content Negotiation (for JSON)
  install(ContentNegotiation) {
       json() // Use kotlinx-json for JSON serialization/deserialization
  // 4.2 Define Routes (Endpoints)
  routing {
      // 4.2.1 GET /products (Get All Products)
      get("/products") {
          call.respond(products) // Respond with the product list (as
JSON)
```

```
fun Application.module() {
  install(ContentNegotiation) {
    json() // Set up JSON serialization
  routing {
    route("/products") {
       // GET endpoint to fetch all
products
         if (products.isEmpty()) {
call.respond(HttpStatusCode.NoContent,
"No products found")
         } else {
            call.respond(products)
       // POST endpoint to add a new
product
       post {
         val product =
call.receive<Product>()
         products.add(product)
call.respond(HttpStatusCode.Created,
product)
```

Step 4: Run the Application

Now, let's create the server and run the Ktor application:

fun main() {

```
// 4.2.2 GET /products/{id} (Get Product by ID)
      get("/products/{id}") {
          val id = call.parameters["id"]?.toIntOrNull() // Get ID from
path
          if (id == null) {
              call.respond(HttpStatusCode.BadRequest, "Invalid product
ID")
              return@get // Exit if ID is not an integer
          val product = products.find { it.id == id } // Find product by
ID
          if (product != null) {
              call.respond(product) // Respond with product (as JSON)
          } else {
              call.respond(HttpStatusCode.NotFound, "Product not found")
      // 4.2.3 POST /products (Create a New Product)
      post("/products") {
          val product = call.receive<Product>() // Get Product from JSON
```

```
embeddedServer(Netty, port = 8080,
module = Application::module).start(wait =
true)
}
```

- **GET /products**: Will return the list of all products as JSON.
- POST /products: Will accept a product in JSON format and add it to the list of products.

Step 5: Test the API

GET Request:

curl -X GET http://localhost:8080/products

Response (if products exist):

POST Request:

curl -X POST http://localhost:8080/products -H "Content-Type: application/json" -d '{"id":3,"name":"Tablet","price":499.99}'

Response:

```
{

"id": 3,

"name": "Tablet".
```

```
in request body
          // Validation: Check for valid data
          if (product.id <= 0 || product.name.isBlank() || product.price
<= 0) {
              call.respond(
                  HttpStatusCode.BadRequest,
                   "Invalid product data. ID and price must be positive,
name cannot be blank."
              return@post
          // Check for duplicate IDs
          if (products.any { it.id == product.id }) {
              call.respond(HttpStatusCode.Conflict, "Product with this ID
already exists.")
              return@post
          products.add(product) // Add the new product
          call.respond(HttpStatusCode.Created, product) // Respond with
```

Function Summarization (Python)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	<pre>Write a brief summary of the function: def reverse_words (sentence) : return ' '. join (sentence . split () [:: -1])</pre>	The function reverse_words(sentence) takes a string sentence as input and returns a new string where the words in the sentence are reversed in order. It does so by splitting the sentence into words, reversing the order of the words, and then joining them back together with spaces.	The reverse_words function takes a string (sentence) as input, splits it into individual words, reverses the order of those words, and then joins the reversed words back into a single string, which it returns. Essentially, it reverses the order of words in a sentence.
Few-shot	<pre>Example 1: def is_even(n): if n % 2 == 0: return True else: return False This method determines whether a number is even. Example 2: def is_prime(n): if n <= 1: return False</pre>	The function reverse_words(sentence) takes a string sentence and returns a new string with the words reversed in order. It splits the sentence into words, reverses the list of words, and then joins them back together with spaces.	The reverse_words function takes a sentence (string) as input and returns a new string with the order of the words reversed.

```
for i in range(2, n):
    if n%i == 0:
        return False
    return True
This method determines if a number is prime.

Write a brief summary of the function:
    def reverse_words ( sentence ) :
    return ' '. join ( sentence . split () [:: -1])
```

Prompt from Code Comments (Python)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	<pre>Write a prompt that could generate the code: # This function checks if a number is prime def is_prime (n) : if n <= 1: return False for i in range (2 , int(n **0.5) +1) : if n % i == 0: return False return True</pre>	Prompt: "Write a Python function named is_prime that takes a number n as input and returns True if the number is prime, and False otherwise. The function should check if the number is less than or equal to 1 and return False in that case. For numbers greater than 1, the function should check divisibility only up to the square root of n to optimize the process."	"Write a Python function called is_prime that takes an integer n as input and returns True if n is a prime number, and False otherwise. Optimize the primality check by only iterating up to the square root of n."
Few-shot	<pre>Example 1: Code: # This function checks if a number is even. def is_even(n): if n % 2 == 0: return True else: return False Prompt: Write a function that checks if a number is even.</pre>	Prompt: "Write a function that checks if a number is prime. The function should return True if the number is prime and False otherwise. It should handle edge cases like numbers less than or equal to 1 and efficiently check divisibility by iterating from 2 up to the square root of the number."	"Write a Python function called is_prime that takes an integer n as input and returns True if n is a prime number, and False otherwise. Optimize the primality check by only iterating up to the square root of n."

```
Example 2:
    Code:
        # This function checks if a number is odd.
    def is_odd(n):
        if n % 2 == 0:
            return Halse
        else:
            return True

Prompt: Write a function that checks if a number is odd.

Write a prompt that could generate the code:
        # This function checks if a number is prime def is prime (n):
        if n <= 1:
        return False
        for i in range (2 , int(n **0.5) +1):
            if n % i == 0:
                 return False
        return False
        return False
```

Fixing Factorial Bug (Python)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Chain-of-thought	Fix the bug when input is 0. Do it step by step: def factorial (n): result = 1 for i in range (1 , n): result *= i	Bug Identification: The bug occurs when the input is θ . In the given function, the loop starts at 1, and when $n=\theta$, the loop will not run. This leads to an incorrect result, as the factorial of θ is defined to be 1. Step-by-Step Fix:	Step 1: Identify the bug The original code calculates the factorial of a number n. The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. • For n > 0, it works correctly. For example, 5! = 5 * 4 * 3 * 2 * 1 = 120. • However, when n = 0, the loop for i in range(1, n) does not execute because range(1, 0) produces an empty sequence. As a result, the

return result

1. Identify the problem:

The factorial of \emptyset is 1, but the current function doesn't account for \emptyset . It should handle this case explicitly.

2. Add a base case for 0:

We need to check if n == 0 before starting the loop and return 1 for that case.

3. Update the loop range:

The loop currently runs from 1 to n-1. If we fix the base case, the loop can stay as is, but ensure that it correctly calculates for other numbers greater than 0.

Fixed Code:

```
def factorial(n):
    if n == 0:
        return 1 # Factorial of 0 is 1
    result = 1
    for i in range(1, n + 1): # Loop from 1 to n,
inclusive
        result *= i
    return result
```

- function returns the initial value of result, which is 1. This is the correct value for O! (0 factorial is defined as 1).
- When n is negative, the loop also does not execute and returns 1, which is incorrect. Factorial is not defined for negative numbers.

So, the code actually returns the correct result for 0, but it should also include a check to disallow negative numbers.

Step 2: Handle the base case (n = 0) and invalid input (n < 0)

We need to explicitly handle the case where \underline{n} is 0 and the case where \underline{n} is negative.

Step 3: Implement the corrected code

Here's the corrected code:

```
Python
```

```
def factorial(n):
```

Calculates the factorial of a non-negative integer.

Args:

n: A non-negative integer.

Returns:

result *= i

```
The factorial of n, or 1 if n is 0.

Raises ValueError if n is negative.

"""

if n < 0:
    raise ValueError("Factorial is not defined for negative numbers")

if n == 0:
    return 1

result = 1

for i in range(1, n + 1):
```

Self-consistency Testing several different fixes, and finding the most efficient one: fix the bug when input is 0. def factorial (n) :

Bug Description:

The function calculates the factorial of a number n, but it doesn't handle the case when n=0. The factorial of 0 is defined as 1, but the current code doesn't account for that, so it returns an incorrect result when n=0.

Step 1: Fix 1 - Add a check for 0 before the loop

This fix will explicitly handle n = 0 before entering the loop.

```
def factorial(n):
    if n == 0:
        return 1 # Factorial of 0 is 1
    result = 1
    for i in range(1, n + 1): # Loop from 1 to n
        result *= i
    return result
```

Explanation:

- If n = 0, we return 1 immediately without entering the loop.
- If n > 0, the loop works as expected, calculating the factorial from 1 to n.

Step 2: Fix 2 - Handle 0 inside the loop

Another fix could be to handle the $\ensuremath{0}$ input within the loop itself.

def factorial(n):

return result

Understanding the Issue

The original code has a couple of issues:

- Incorrect loop for n = 0: When n is 0, the range(1, n) produces an empty sequence, so the loop doesn't execute, and the function correctly returns the initial value of result (which is 1). While it works, it's not the most explicit way to handle the base case.
- 2. **Missing input validation:** The function doesn't handle negative input, for which the factorial is undefined.

Fixes and Efficiency Considerations

Here are a few different approaches to fix the bug and improve the code, along with discussions of their efficiency:

1. Explicitly Handle the Base Case and Validate Input (Most Efficient)

```
Python
```

```
def factorial_v1(n):
    """Calculates the factorial of a non-negative integer."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0:
        return 1
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result</pre>
```

• Explanation:

- We explicitly check for n < 0 and raise a ValueError.
- We explicitly check for n == 0 and return 1.

```
result = 1
for i in range(1, n):
    result *= i
    return result if n != 0 else 1 # Handle the special
case for 0
```

Explanation:

 The loop works the same way for n > 0, and the special case for n = 0 is handled at the return statement by checking if n
 == 0.

Step 3: Fix 3 - Use math.factorial() for efficiency

We could use the math.factorial() function, which already handles all edge cases like n = 0 efficiently.

import math

def factorial(n):
 return math.factorial(n)

Explanation:

 The built-in math.factorial() function is designed to handle the factorial calculation efficiently, including for n = 0.

Step 4: Performance Consideration

• Fix 1: This approach directly checks if n =

- \circ The loop now correctly iterates from 1 to n (inclusive) for n > 0.
- **Efficiency:** This is generally the most efficient approach. The conditional checks at the beginning are very fast, and the loop iterates the minimum number of times necessary.

2. Slightly More Concise (But Equally Efficient)

```
Python

def factorial_v2(n):
    """Calculates the factorial of a non-negative integer."""
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    result = 1
    for i in range(1, n + 1): # Corrected loop range
        result *= i
    return result
```

- **Explanation:** This version combines the initialization of result with the main logic and relies on the loop not executing when n is 0.
- **Efficiency:** This is equally efficient to v1. The conditional check is still very fast, and the loop behavior is the same.

3. Using math.factorial (For Comparison)

Python

import math

def factorial v3(n):

"""Calculates the factorial of a non-negative integer using math.factorial.""" if n < 0:

raise ValueError("Factorial is not defined for negative numbers") return math.factorial(n)

- 0, which is an explicit and simple check. It's fast and works for all inputs, including edge cases like 0.
- Fix 2: This approach handles 0 after the loop, but it still works fine. However, it involves an extra conditional check in the return statement.
- Fix 3: Using math.factorial() is likely the most efficient for large inputs, as it is highly optimized and implemented in C.
 For small numbers, it's not noticeably faster but is a more robust solution.

Most Efficient Fix:

- Fix 3 (Using math.factorial()) is the most efficient overall, as it leverages a highly optimized built-in function.
- However, for small values of n or for simplicity, Fix 1 (handling 0 explicitly before the loop) is a very clean and efficient solution for most cases.

- **Explanation:** Python's math module provides a built-in factorial function.
- Efficiency: The math.factorial function is generally implemented in C and is highly optimized. It's likely to be the fastest of all the versions, especially for larger values of n.

Why Version 1 and 2 are Good Choices

Versions 1 and 2 are good choices because they are:

- Correct: They handle the base case (n = 0) and invalid input (n < 0) correctly.
- **Efficient:** They use a straightforward loop that performs the necessary multiplications without unnecessary overhead.
- Readable: The code is easy to understand and follow.

When to Use math.factorial

If you need the absolute best performance, especially for large values of n, use math.factorial. It's the most efficient way to calculate factorials in Python. However, for educational purposes or when you want to implement the logic yourself, versions 1 or 2 are perfectly fine.

The Most Efficient Fix

The most efficient fix, in terms of raw speed, is to use Python's built-in math.factorial (version 3). If you're implementing the factorial function yourself, versions 1 and 2 are very close in efficiency and are both good choices.

```
Prompt method:
                Prompt:
                                      GPT-4 output:
                                                                              Gemini output:
Zero-shot
                Implement node
                                      #include <stdio.h>
                                                                              #include <stdio.h>
                deletion by value:
                                      #include <stdlib.h>
                                                                              #include <stdlib.h>
                 struct Node {
                                      struct Node {
                  int data ;
                                        int data:
                                                                              struct Node {
                 struct Node *
                                        struct Node* next;
                 next ;
                                      // Function to delete the node by value
                 void deleteNode
                                      (key)
                  struct Node
                                      void deleteNode(struct Node** head, int
                                      key) {
                 ** head , int
                                                                              void deleteNode(struct Node **head, int key) {
                                        // Step 1: Check if the list is empty
                 key ) {
                                        if (*head == NULL) {
                                                                                 struct Node *current = *head;
                 // TODO :
                                          printf("List is empty.\n");
                                          return;
                 Implement node
                 deletion
                                        // Step 2: If the node to be deleted is
                                      the head node
                                                                                 if (*head == NULL) {
                                        if ((*head)->data == key) {
                                          struct Node* temp = *head;
                                          *head = (*head)->next; // Update the
                                      head to the next node
                                          free(temp); // Free the memory of
                                      the deleted node
                                          return:
                                                                                 if (current != NULL && current->data == key) {
                                                                                      *head = current->next; // Change head to the next node
                                        // Step 3: Traverse the list to find the
                                                                                      free(current);
                                      node to be deleted
                                        struct Node* current = *head:
                                        while (current->next != NULL &&
                                      current->next->data != key) {
                                          current = current->next: // Move to
                                      the next node
                                                                                 while (current != NULL && current->data != key) {
                                        // Step 4: If the node with the given key
                                      was found
                                        if (current->next != NULL) {
                                                                                      current = current->next;
```

```
struct Node* temp = current->next;
     current->next = current->next->next:
// Skip the node to be deleted
     free(temp); // Free the memory of
the deleted node
  } else {
     printf("Node with value %d not
found.\n", key); // If the key is not found
 }
// Utility function to create a new node
struct Node* newNode(int data) {
  struct Node* node = (struct
Node*)malloc(sizeof(struct Node));
  node->data = data;
  node->next = NULL:
  return node:
// Utility function to print the linked list
void printList(struct Node* head) {
  while (head != NULL) {
     printf("%d -> ", head->data);
     head = head->next;
  printf("NULL\n");
int main() {
  // Create a simple linked list for testing
  struct Node* head = newNode(10);
  head->next = newNode(20);
  head->next->next = newNode(30);
  head->next->next->next =
newNode(40);
  printf("Original List: ");
  printList(head);
  deleteNode(&head, 20); // Delete node
with value 20
  printf("List after deleting 20: ");
  printList(head);
```

```
if (current == NULL) {
  prev->next = current->next;
  free(current);
void printList(struct Node *node) {
  while (node != NULL) {
      node = node->next;
  printf("\n");
struct Node *newNode(int data) {
  struct Node *node = (struct Node *)malloc(sizeof(struct Node));
  node->data = data;
  node->next = NULL;
  return node;
```

```
deleteNode(&head, 10); // Delete node
with value 10
    printf("List after deleting 10: ");
    printList(head);

    deleteNode(&head, 50); // Attempt to
delete node with value 50 (not found)
    printf("List after trying to delete 50: ");
    printList(head);

    return 0;
}
```

```
int main() {
  struct Node *head = newNode(1);
  head->next = newNode(2);
  head->next->next = newNode(3);
  head->next->next->next = newNode(4);
  head->next->next->next = newNode(5);
  printList(head);
  deleteNode(&head, 3);
  printList(head); // Expected: 1 2 4 5
  deleteNode(&head, 1);
  printList(head); // Expected: 2 4 5
  deleteNode(&head, 5);
  printf("List after deleting 5: ");
  printList(head); // Expected: 2 4
```

```
deleteNode(&head, 6);
                                                                        printf("List after deleting 6: ");
                                                                        printList(head); // Expected: 2 4 (no change)
                                                                        struct Node *onlyHead = newNode(10);
                                                                        deleteNode(&onlyHead, 10);
Chain-of-thoug
               Implement node
                                                                     #include <stdio.h>
ht
               deletion by value. Do
                                 Step-by-Step Explanation
```

it step-by-step:

struct Node { int data ; struct Node * next ; void deleteNode struct Node ** head , int key) { Implement node deletion

- 1. Check if the list is empty:
 - o If the list is empty (i.e., *head == NULL). there's nothing to delete, so we return early.
- 2. Delete the head node:
 - If the node to be deleted is the head node, update the head pointer to point to the next node.
- 3. Traverse the list:
 - o If the node to be

```
#include <stdlib.h>
struct Node {
  struct Node *next;
void deleteNode(struct Node **head, int key) {
nothing to do.
  if (*head == NULL) {
```

deleted is not the head, traverse the list to find the node that contains the value (key). You will need to keep track of the previous node to adjust the next pointer correctly.

4. Delete the node:

- Once the node with the matching value is found, change the next pointer of the previous node to skip the node that is being deleted
- Free the memory allocated for the node that was deleted.

Here's the step-by-step code:

Step 1: Node Definition

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
    struct Node* next;
};
```

Step 2: Function to Delete Node by Value

void deleteNode(struct Node** head, int key) {

// Step 1: Check if the list is empty

```
if (current != NULL && current->data == key) {
    *head = current->next; // Change the head pointer to point to the
    free(current);
while (current != NULL && current->data != key) {
    current = current->next; // Move 'current' to the next node
if (current == NULL) {
prev->next = current->next; // Make the previous node point to the node
```

```
if (*head == NULL) {
     printf("List is empty.\n");
     return:
  // Step 2: If the node to be deleted is
the head node
  if ((*head)->data == key) {
     struct Node* temp = *head;
     *head = (*head)->next; // Move the
head to the next node
     free(temp); // Free the memory of
the deleted node
     return;
  // Step 3: Traverse the list to find the
node to be deleted
  struct Node* current = *head;
  while (current->next != NULL &&
current->next->data != key) {
     current = current->next: // Move to
the next node
  // Step 4: If the node with the given key
was found
  if (current->next != NULL) {
     struct Node* temp = current->next;
     current->next = current->next->next;
// Skip the node to be deleted
     free(temp); // Free the memory of
the deleted node
 } else {
     printf("Node with value %d not
found.\n", key); // If the key is not found
Step 3: Utility Functions to Create
Nodes and Print the List
```

// Utility function to create a new node

```
deleted.
void printList(struct Node *node) {
  printf("\n");
struct Node *newNode(int data) {
  struct Node *node = (struct Node *)malloc(sizeof(struct Node));
  node->next = NULL;
int main() {
  struct Node *head = newNode(1);
  head->next = newNode(2);
  head->next->next = newNode(3);
  head->next->next->next = newNode(4);
  head->next->next->next = newNode(5);
```

```
Node*)malloc(sizeof(struct Node));
  node->data = data;
  node->next = NULL;
  return node:
// Utility function to print the linked list
void printList(struct Node* head) {
  while (head != NULL) {
     printf("%d -> ", head->data);
     head = head->next:
  printf("NULL\n");
Step 4: Main Function for Testing
int main() {
  // Create a simple linked list for testing
  struct Node* head = newNode(10);
  head->next = newNode(20);
  head->next->next = newNode(30);
  head->next->next->next =
newNode(40);
  printf("Original List: ");
  printList(head);
  deleteNode(&head, 20); // Delete node
with value 20
  printf("List after deleting 20: ");
  printList(head):
  deleteNode(&head, 10); // Delete node
with value 10
  printf("List after deleting 10: ");
  printList(head);
  deleteNode(&head, 50); // Attempt to
delete node with value 50 (not found)
  printf("List after trying to delete 50: ");
```

struct Node* newNode(int data) {
 struct Node* node = (struct

```
printf("Original List: ");
printList(head);
deleteNode(&head, 3);
printf("List after deleting 3: ");
printList(head); // Expected: 1 2 4 5
deleteNode(&head, 1);
printf("List after deleting 1: ");
printList(head); // Expected: 2 4 5
deleteNode(&head, 5);
printList(head); // Expected: 2 4
deleteNode(&head, 6);
printf("List after deleting 6: ");
printList(head); // Expected: 2 4 (no change)
struct Node *onlyHead = newNode(10);
deleteNode(&onlyHead, 10);
printf("List after deleting the only node (10): ");
printList(onlyHead); //Expected to print nothing
```

```
printList(head);
return 0;
}
```

Recursive Function Completion (Python)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	Complete the recursive function for Fibonacci: def fibonacci (n): # TODO: Base cases and recursive call pass	def fibonacci(n): # Base cases if n == 0: return 0 elif n == 1: return 1 # Recursive case return fibonacci(n - 1) + fibonacci(n - 2)	<pre>def fibonacci(n): """ Calculates the nth Fibonacci number using recursion. Args: n: A non-negative integer. Returns: The nth Fibonacci number. Raises: ValueError: If n is negative. """ if n < 0: raise ValueError("Fibonacci is not defined for negative numbers") if n == 0: return 0 elif n == 1:</pre>

```
for i in range(10):
                                                                                                           print(f"Fibonacci({i}) = {fibonacci(i)}")
                                                                                                          try:
                                                                                                           print(fibonacci(-1))
                                                                                                          except ValueError as e:
Chain-of-thoug
                Complete the
                                                                                                          def fibonacci(n):
                recursive function for
ht
                                      Step 1: Base Case
                Fibonacci. Do it step
                by step:
                                      The base case is essential in a recursive function to stop the recursion.
                def fibonacci (
                                      The Fibonacci sequence is defined as:
                n ) :

    fibonacci(0) should return 0.

    fibonacci(1) should return 1.

                                     Step 2: Recursive Case
                                      For all other values of n, the Fibonacci number is the sum of the two
                                      preceding Fibonacci numbers:
                                          • fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
                                             for n > 1.
```

Step 3: Combining Both

Now, let's implement the recursive function using these two parts: the base case and the recursive case.

Code Implementation

```
def fibonacci(n):
    # Step 1: Base cases
    if n == 0:
        return 0
    elif n == 1:
        return 1

# Step 2: Recursive case
    return fibonacci(n - 1) + fibonacci(n - 2)
```

Explanation:

1. Base Cases:

- \circ When n == 0, return 0 (the 0th Fibonacci number).
- When n == 1, return 1 (the 1st Fibonacci number).

2. Recursive Case:

 For any value of n > 1, the function recursively calls itself for fibonacci(n-1) and fibonacci(n-2) and adds their results.

```
raise ValueError("Fibonacci is not defined for
  return fibonacci(n - 1) + fibonacci(n - 2) #
for i in range(10):
print(f"Fibonacci({i}) = {fibonacci(i)}")
print(fibonacci(-1))
```

Constructor Completion (Python)

Prompt method: Prompt: GPT-4 output: Gemini output:

```
Zero-shot

Complete the class constructor: class Person:

def __init__ ( self ):

# TODO:

Add name ,

age , and

optional

email
```

```
class Person:
    def __init__(self, name, age,
    email=None):
        self.name = name  # Required:
    name of the person
        self.age = age  # Required:
    age of the person
        self.email = email  # Optional:
    email of the person (default is None)
```

```
class Person:
def init (self, name, age, email=None):
  Constructs a Person object.
  if not isinstance(name, str):
    raise TypeError("Name must be a string")
  if not isinstance(age, int):
    raise TypeError("Age must be an integer")
  if age < 0:
    raise ValueError("Age cannot be negative")
  if email is not None and not isinstance (email, str):
      raise TypeError("Email must be a string or None")
```

```
if self.email:
                                                                             return f"{self.name}, {self.age} years old"
                                                                        def greet(self):
                                                                          return f"Hello, my name is {self.name} and I am {self.age} years old."
Chain-of-thought
                 Complete the
                                                                       class Person:
                 class constructor.
                                   Step 1: Define Required Parameters
                 Do it step by
                                   (name and age)
                 step:
                 class Person
                                       • The name and age are
                                           required fields, meaning they
                                           must be passed in when
                                           creating a Person object.
                                                                          Constructs a Person object.
                                   Step 2: Add an Optional email
                 optional
                                       • The email is optional,
                                           meaning it might or might not
                                           be provided. To make it
                                           optional, we can use the
                                           default value None for email
                                           in the constructor.
                                   Step 3: Initialize the Object's
                                                                          if not isinstance(name, str):
                                   Attributes
                                                                             raise TypeError("Name must be a string")
                                   We will now initialize the attributes
```

if not isinstance(age, int) or age < 0:

name, age, and email inside the

constructor using self.

Code Implementation (Step-by-Step):

Step 1: Add name and age as required parameters:

class Person:
 def __init__(self, name, age):
 self.name = name
 self.age = age

 Here, name and age are initialized as required attributes

Step 2: Add email as an optional parameter:

class Person:
 def __init__(self, name, age,
 email=None):
 self.name = name
 self.age = age
 self.email = email # Optional,
 default is None

 The email parameter is optional and defaults to None if no value is provided.

Final Code:

class Person:

```
raise ValueError("Age must be a non-negative integer")
if email is not None and not isinstance (email, str):
 raise TypeError("Email must be a string or None")
self.email = email
 if self.email:
     return f"{self.name}, {self.age} years old, Email: {self.email}"
     return f"{self.name}, {self.age} years old"
 return f"Hello, my name is {self.name} and I am {self.age} years old."
```

```
def __init__(self, name, age, email=None):
    self.name = name # Required:
    name of the person
    self.age = age # Required:
    age of the person
    self.email = email # Optional:
    email of the person (default is None)
```

Binary Search Completion (Java)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	Complete the binary search implementation: public int binarySearch (int [] arr , int target) { int left = 0 , right = arr . length - 1; while (left <= right) { int mid = (left + right) / 2; // TODO : Compare and adjust bounds }	<pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = (left + right) / 2; // Compare and adjust bounds if (arr[mid] == target) { return mid; // Found the target at index mid } else if (arr[mid] < target) { left = mid + 1; // Target is in the right half } else { right = mid - 1; // Target is in the left half } } return -1; // Target not found }</pre>	<pre>public class BinarySearch { /** * Performs a binary search on a sorted integer array. * * @param arr The sorted integer array to search. * @param target The integer value to search for. * @return The index of the target value in the array if found, otherwise -1. */ public static int binarySearch(int[] arr, int target) { int left = 0; int right = arr.length - 1; while (left <= right) { int mid = left + (right - left) / 2; // Prevent potential overflow</pre>

```
if (arr[mid] == target) {
                                                                            } else if (arr[mid] < target) {</pre>
                                                                    public static void main(String[] args) {
                                                                        int[] sortedArray = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
                                                                        int targetValue = 23;
                                                                        int resultIndex = binarySearch(sortedArray, targetValue);
                                                                            System.out.println("Target " + targetValue + " found at index " +
                                                                 resultIndex);
                                                                            System.out.println("Target " + targetValue + " not found in the
                                                                 array");
                                                                public class BinarySearch {
Chain-of-though
              Complete the binary
              search
                                  Step 1: Initialize and Setup
              implementation. Do it
              step by step:
```

```
public int
binarySearch (
int [] arr , int
target ) {
  int left = 0 ,
  right = arr .
length - 1;
  while ( left <=
  right ) {
    int mid = (
  left + right ) /
  2;
    // TODO :
Compare and
adjust bounds
  }
  return -1;
}</pre>
```

We start with the basic setup:

- left = 0: This is the starting index of the array.
- right = arr.length 1: This is the last index of the array.
- We calculate the midpoint using int mid = (left + right) / 2;.

Step 2: Compare the Midpoint with the Target

Now, we need to compare the value at the midpoint (arr[mid]) with the target:

- If arr[mid] == target, we found the target at index mid, so we should return mid.
- If arr[mid] < target, the target must be on the right side, so we adjust left to mid + 1.
- If arr[mid] > target, the target must be on the left side, so we adjust right to mid - 1.

Step 3: Loop Continuation and Exit

```
-1.
  public static int binarySearch(int[] arr, int target) {
array.
prevents overflow.
           int mid = left + (right - left) / 2;
overflow
           if (arr[mid] == target) {
               return mid; // Step 4a: Target found! Return the index.
           } else if (arr[mid] < target) {</pre>
```

```
The loop continues as long as left
<= right. Once the loop ends, it</p>
means the target is not found, so we
return -1.
Complete Code Implementation:
public int binarySearch(int[] arr, int
target) {
 int left = 0, right = arr.length - 1;
  while (left <= right) {
    int mid = (left + right) / 2; //
Calculate the midpoint
                                        public static void main(String[] args) {
    // Compare the target with
arr[mid]
    if (arr[mid] == target) {
                                             int targetValue = 23;
      return mid; // Target found,
return the index
    } else if (arr[mid] < target) {
                                             int resultIndex = binarySearch(sortedArray, targetValue);
      left = mid + 1; // Target is in
the right half, move left bound
    } else {
                                             if (resultIndex != -1) {
      right = mid - 1; // Target is in
the left half, move right bound
                                                  System.out.println("Target " + targetValue + " found at index " +
                                    resultIndex);
  return -1; // Target not found
                                                  System.out.println("Target " + targetValue + " not found in the
                                    array");
```

Self-Consistency Bug Fixing (C++)

Prompt method: Prompt: GPT-4 output: Gemini output:

Resolve inconsistency
between function name
and logic:
// Supposed to
return true if x
is even
bool isOdd (int x
) {
 return x % 2 ==
0; // Logic
 contradicts
 function name
}

// Correct function name and logic to return true if x is odd bool isOdd(int x) {

return x % 2 != 0; // Return true if x is odd

// Returns true if x is even
bool isEven(int x) {
 return x % 2 == 0;
}

// Solution 2: Correct the logic
// Returns true if x is odd
bool isOdd(int x) {
 return x % 2 != 0;
}

Self-consistency

and briefly summarize the function based on those examples. Then, determine whether there is an inconsistency between function name and logic. Resolve the inconsistency if present: // Supposed to return true if x is even bool isOdd (int x) { return x % 2 == 0; // Logic contradicts function name

Provide several input and output examples.

Step 1: Review the current code and examples

The function isOdd is currently designed to return true if a number is **even**, based on the logic x % 2 = 0. However, the function name isOdd suggests that it should return true if a number is **odd**, which creates an inconsistency between the function name and its logic.

Step 2: Input and Output Examples

Example 1:

isOdd(4);

- Expected Output: false (since 4 is even)
- Actual Output: true (because 4 % 2 == 0)

```
bool isOdd(int x) {
  return x % 2 == 0; // Logic contradicts function name
}

#include <iostream>

int main() {
  // Input/Output Examples
  std::cout << "isOdd(0): " << (isOdd(0) ? "true" : "false") <<
  std::endl; // Output: true
  std::cout << "isOdd(1): " << (isOdd(1) ? "true" : "false") <<
  std::endl; // Output: false
  std::cout << "isOdd(2): " << (isOdd(2) ? "true" : "false") <<
  std::endl; // Output: true
  std::cout << "isOdd(3): " << (isOdd(3) ? "true" : "false") <<</pre>
```

Example 2:

isOdd(7);

- Expected Output: true (since 7 is odd)
- Actual Output: false (because 7 % 2 == 1)

Example 3:

isOdd(0);

- Expected Output: false (since 0 is even)
- Actual Output: true (because 0 % 2 ==
 0)

Step 3: Summary of Function Behavior

Based on the examples, the function isOdd is currently behaving like a function that checks if a number is **even**, not odd. Specifically:

- It returns true when the number is **even**.
- It returns false when the number is odd.

However, the function name isOdd contradicts this behavior, as it implies the function should return true for odd numbers.

Step 4: Inconsistency

There is indeed an **inconsistency** between the

```
std::endl; // Output: false
<< std::endl; // Output: true
std::cout << "isOdd(-3): " << (isOdd(-3) ? "true": "false")</pre>
<< std::endl; // Output: false
 std::cout << "\nFunction Summary:\n";</pre>
 std::cout << "The isOdd function returns true if the input</pre>
std::cout << "and false if x is odd. It does this by checking</pre>
std::cout << "The function should be named isEven, or its</pre>
 std::cout << "Corrected isOdd function:\n";</pre>
```

function name is0dd and its logic:

- Name isOdd implies the function should return true for odd numbers.
- Logic (x % 2 == 0) checks if a number is even, so it returns true for even numbers.

Step 5: Resolution

To resolve this inconsistency, we need to:

- Change the function name to something that reflects its actual behavior, such as isEven.
- 2. Or, we could **update the logic** to make the function behave according to its name is0dd.

Option 1: Change the Function Name to isEven

```
// Correct name and logic for even check
bool isEven(int x) {
  return x % 2 == 0; // Return true if x is even
}
```

Option 2: Change the Logic to Match is0dd

```
// Keep the function name isOdd, but update the logic
to check for odd numbers
bool isOdd(int x) {
   return x % 2 != 0; // Return true if x is odd
}
```

```
bool isOddCorrected(int x) {
std::cout << "isOdd(0): " << (isOddCorrected(0) ? "true":</pre>
"false") << std::endl;
 std::cout << "isOdd(1): " << (isOddCorrected(1) ? "true":</pre>
"false") << std::endl;
 std::cout << "isOdd(2): " << (isOddCorrected(2) ? "true":</pre>
"false") << std::endl;
 std::cout << "isOdd(3): " << (isOddCorrected(3) ? "true":</pre>
"false") << std::endl;
"false") << std::endl;
 std::cout << "isOdd(-3): " << (isOddCorrected(-3) ? "true":</pre>
"false") << std::endl;
function):\n";
 std::cout << "Renamed function:\n";</pre>
 std::cout << "bool isEven(int x) {\n";</pre>
```

Prompt method: Prompt: GPT-4 output:	Gemini output:
Prompt-chaining Identify the bug in this code (do not fix it): // Supposed to return true if x is even bool isOdd (int x) { return x % 2	Gemini output: The bug is that the function isOdd incorrectly returns true if x is even and false if x is odd. The logic contradicts the function's name. -> // Solution 1: Rename the function to isEven bool isEven(int x) { return x % 2 == 0; } // Solution 2: Correct the logic to check for odd numbers bool isOdd(int x) { return x % 2 != 0; }

Chain-of-thought

Identify and fix the bug. Do it **step by step**:

// Supposed to
return true if
x is even
bool isOdd (int
x) {
return x % 2
== 0; // Logic
contradicts
function name
}

Step 1: Identify the Bug

The function name is0dd suggests that it is supposed to return true if x is odd. However, the logic x % 2 == 0 checks if x is **even**, returning true for even numbers. This creates a contradiction between the function name and its logic.

Step 2: Analyze the Logic

- The **logic** x % 2 == 0 checks whether a number is even.
- The function name isOdd implies the function should check if a number is odd.

Step 3: Fixing the Bug

There are two ways to resolve this issue:

Option 1: Change the Function Name to isEven

If the intent of the function is to check whether x is even, the **function name** should be changed to isEven. The logic of checking for even numbers (x = 0) is correct.

// Corrected: Change function name to isEven to
match the logic
bool isEven(int x) {
 return x % 2 == 0; // Return true if x is even
}

Option 2: Change the Logic to Match isOdd

Step 1: Identify the bug

The code is intended to return true if x is even, but the function is named isOdd. The logic within the function checks for evenness (x % 2 == 0) which contradicts the function name.

Step 2: Determine the intended behavior

There are two ways to interpret the intended behavior, leading to two possible fixes:

- **Interpretation 1:** The function is *supposed* to return true if x is even. In this case, the name is wrong.
- Interpretation 2: The function is *supposed* to return true if x is odd. In this case, the logic is wrong.

Step 3: Fix the code (Option 1: Rename the function)

If the intended behavior is to check for even numbers, we should rename the function:

```
C++

bool isEven(int x) {
  return x % 2 == 0;
}
```

Step 4: Fix the code (Option 2: Correct the logic)

If the intended behavior is to check for odd numbers, we should correct the logic:

```
C++

bool isOdd(int x) {
    return x % 2 != 0;
}
```

If the function is supposed to check whether x is odd, the **logic** should be updated to check if the number is odd, which is done by using x % 2 ! = 0.

// Corrected: Change the logic to check if x is odd bool isOdd(int x) { return x % 2 != 0; // Return true if x is odd

Step 5: Choose the best fix

The "best" fix depends on the context of how this function is used. If other parts of the code rely on it actually checking for even numbers, then renaming it to is Even is the least disruptive change. If the surrounding code expects it to check for odd numbers, then correcting the logic is better.

Without additional context, it's impossible to say definitively which fix is "best." Both are valid ways to resolve the inconsistency. I'll provide both for completeness.

Summary Decomposition (C++)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	Decompose the high-level comment/summary into logical steps: // Function that validates an input , calculates square , and returns result	High-Level Comment:	Here's a breakdown of the logical steps the process function performs:
	<pre>int process (int x) { if (x < 0) return -1; return x * x; }</pre>	Step-by-Step Decomposition: 1. Validate the input:	Check for negative input: It checks if the input integer x is less than 0.
	J	 The function checks if the input x is valid. In this case, the validation is done by checking if x is negative. If x is negative (x < 0), the function returns -1 to indicate an invalid input. 	2. Handle negative input: If x is negative, the function immediately
		Calculate the square: If the input x is valid (i.e., not negative), the function proceeds to calculate the	returns -1. 3. Calculate the square: If x is not

		square of x. The square of x is calculated by multiplying x by itself (x $*$ x). 3. Return the result :	negative (i.e., it's zero or positive), the function calculates the
		 The function returns the result of the square of x if the input was valid. Otherwise, it returns -1 for invalid input. 	square of x (x * x). 4. Return the result: The function returns the
		Logical Steps in Code:	calculated square (if x was not
		1. Check if x is negative:	negative) or -1 (if x was negative).
		○ If x is less than 0, return -1.	
		2. Otherwise, calculate the square of x:	
		\circ If x is valid (non-negative), compute x * x and return the result.	
		Code with Clear Logical Steps:	
		<pre>int process(int x) { // Step 1: Validate input (check if x is negative) if (x < 0) { return -1; // Invalid input (negative) }</pre>	
		// Step 2: Calculate square (if valid input) return x * x; // Return the square of x }	
		This breakdown ensures the function validates the input, computes the square, and then returns the result accordingly.	
Few-shot	Example 1: Code: // Function that returns true	High-Level Comment:	Code:
	if a number is even		// Function that validates an input, calculates square,

```
bool isEven(int number) {
  return number % 2 == 0;
}
```

Steps:

Receives a numerical input, divides by 2, returns true if the remainder is 0 (indicating that the number is even), or returns false if the remainder is not 0 (indicating that the number is not even)

Example 2:

Code:

```
// Function that returns true
if a number is odd
bool isOdd(int number) {
   return number % 2 != 0;
}
```

Steps:

Receives a numerical input, divides by 2, returns true if the remainder is not 0 (indicating that the number is odd), or returns false if the remainder is 0 (indicating that the number is not odd)

Decompose the high-level comment/summary into logical steps:

```
// Function that validates an
input , calculates square ,
and returns result
int process (int x ) {
  if ( x < 0) return -1;
  return x * x;
}</pre>
```

"Function that validates an input, calculates square, and returns result"

Step-by-Step Decomposition:

1. Validate the input:

- The function first checks if the input value x is valid.
- In this case, it validates whether x is negative.
- If x is negative, the function returns -1 (indicating invalid input).

2. Calculate the square:

- \circ If the input is valid (i.e., x is not negative), the function proceeds to calculate the square of x.
- \circ This is done by multiplying x by itself (x * x).

3. Return the result:

- \circ The function returns the result of x * x (the square) if x was valid.
- If x was negative, it returns -1 as an indication of invalid input.

Logical Steps in the Code:

```
int process(int x) {
    // Step 1: Validate input (check if x is negative)
    if (x < 0) {
        return -1; // Return -1 if input is invalid (negative number)
    }

    // Step 2: Calculate square if input is valid
    return x * x; // Return the square of x</pre>
```

```
and returns result
int process(int x) {
  if (x < 0)
    return -1;
  return x * x;
}
```

Steps:

Receives a numerical input x, checks if x is less than 0. If x is less than 0, the function returns -1. Otherwise, the function calculates the square of x (x * x) and returns the result.

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	Complete the function based on intent: def calculate_average (scores) : total = 0 # TODO : Complete to return average pass	def calculate_average(scores): total = sum(scores) # Sum all the scores average = total / len(scores) # Divide the total by the number of scores return average	<pre>def calculate_average(scores): """ Calculates the average of a list of numerical scores. Args: scores: A list of numbers (integers or floats). Returns: The average of the scores (float), or 0 if the list is empty. Raises: TypeError: If the input is not a list. ValueError: If the list contains non-numerical values. """ if not isinstance(scores, list): raise TypeError("Input must be a list.") if not scores: # Check for empty list return 0.0 # Return 0.0 for an empty list, as specified total = 0 for score in scores: if not isinstance(score, (int, float)): raise ValueError("All elements in the list must be numbers.") total += score return float(total) / len(scores)</pre>

Chain-of-thoug

Complete the function based on intent. Do it **step by step**:

def
calculate_average
(scores) :
 total = 0
TODO :
Complete to
return average
pass

Step 1: Sum the scores

 The first step is to calculate the total sum of all the scores in the list. You can use the sum() function to do this efficiently.

Step 2: Count the number of scores

 Next, we need to count how many scores are in the list. This can be done using the len() function, which returns the number of elements in a list.

Step 3: Calculate the average

 The average is calculated by dividing the total sum by the number of elements in the list (i.e., total / len(scores)).

Step 4: Handle edge cases

 If the list is empty (i.e., len(scores)
 == 0), it would be good to return 0 or handle it with an exception to avoid division by zero.

Now, let's implement this step by step:

def calculate_average(scores):
 # Step 1: Sum the scores
 total = sum(scores) # Calculates the sum of
the scores

Step 2: Check if the list is empty to avoid

```
def calculate average(scores):
if not isinstance(scores, list):
  raise TypeError("Input must be a list.")
  if not isinstance(score, (int, float)):
```

```
division by zero
if len(scores) == 0:
return 0 # Return 0 or handle it as needed

# Step 3: Calculate the average
average = total / len(scores)

# Step 4: Return the average
return average

# Step 4: Return the average
return average

# Step 6: Return the calculated average
return average
```

BONUS Full-File Bug Detection and Refactoring (Python)

Prompt method: Prompt:	GPT-4 output:	Gemini output:
Chain-of-thought Analyze the following utility script. Identify any potential logic issues or design flaws Then, refactor it for better readability, correctness, and safety. Do it step by step: # utils .py - Script to parse and summarize numeric CSV files import csv def read_csv (filepath): with open (filepath, 'r') as f: return [row]	o File Handling: The open() function does not have any error handling, so if the file does not exist, it will raise an exception, causing the program to crash. CSV Parsing: The function	<pre># utils.py - Script to parse and summarize numeric CSV files import csv import os from typing import List, Tuple, Optional def read_csv(filepath: str) -> List[List[str]]: """ Reads a CSV file and returns the data as a list of lists, with each inner list representing a row from the CSV. Handles file not found and general IO errors. Args: filepath: The path to the CSV file. Returns:</pre>

```
for row in csv .
reader (f)l
summarize column
( data , index )
for row in data
total = sum(
return total ,
filepath = 'data
filepath )
summarize column
data , 1)
print (" Total
print (" Average
```

there are edge cases where data might not have a header. If the data is malformed (e.g., a single-row CSV), this could lead to incorrect results.

- Error Handling: The float(row[index]) conversion can fail if any value in the column is not numeric. The script does not handle this case, which can cause a runtime exception.
- Empty Data: There is no check for an empty column or a situation where the column might not contain any numeric data, which can result in a division by zero when calculating the average.

3. main() function:

- Data Validity: There is no check to see if the data returned from read_csv() is empty or malformed. If the file is missing or empty, the script will fail in the summarize_column() function.
- Hardcoded File Path: The file path is hardcoded, which makes the script less flexible. It would be better to allow the user to provide the file path as input.
- Potential Index Error: In summarize_column(data, 1), the index 1 is hardcoded, which assumes the relevant column for summarization is

```
already raised by open(), but we document it)
   if not os.path.exists(filepath):
       raise FileNotFoundError(f"File not found at: {filepath}")
       with open(filepath, 'r', encoding='utf-8') as f: #
           reader = csv.reader(f)
           data = list(reader) # Read all rows into a list
   except IOError as e:
       raise IOError(f"Error reading file: {filepath} - {e}")
   except Exception as e:
       raise IOError(f"Unexpected error reading file: {filepath}
def summarize column(data: List[List[str]], index: int) ->
Tuple[float, float]:
   Summarizes a single column from the CSV data, calculating the
total and average
```

always the second column. This may cause an index error if the CSV structure changes.

Refactoring Steps

Step 1: Improve read_csv() with Error Handling

We need to handle errors like file not found and invalid CSV format. Also, ensure that the file is properly closed after reading by using the with statement.

Step 2: Enhance summarize_column() to Validate Data

We need to handle cases where:

- The column is empty or contains non-numeric data.
- The data might have a header or not.

Step 3: Improve the main() Function

We should add checks for empty data, handle file input via command-line arguments, and add better error handling.

Refactored Code

import csv import os

```
Returns (0.0, 0.0) if the column has no valid numeric
     TypeError: If the data is not a list of lists, or if
 if not isinstance(data, list) or not all(isinstance(row,
ist) for row in data):
     raise TypeError("Data must be a list of lists.")
 if not isinstance(index, int):
      raise TypeError("Index must be an integer.")
  if not data: # Handle empty data case
```

```
def read csv(filepath):
  Reads a CSV file and returns a list of rows.
Handles errors like file not found.
  if not os.path.exists(filepath):
     print(f"Error: The file at {filepath} does not
exist.")
     return []
  try:
     with open(filepath, 'r') as f:
       return [row for row in csv.reader(f)]
  except csv.Error as e:
     print(f"Error reading the CSV file: {e}")
     return []
def summarize column(data, index):
  Summarizes a specific column (skipping
header). Returns the total and average of numeric
values.
  if len(data) <= 1:
     print("Error: The file contains no data to
summarize.")
     return 0.0
  values = ∏
  for row in data[1:]: # skip header row
     trv:
       value = float(row[index])
       values.append(value)
     except (ValueError, IndexError):
        print(f"Warning: Non-numeric or missing
value found at row {data.index(row)} in column
{index}. Skipping.")
       continue
  if not values:
     print("Warning: No valid numeric data found in
the column.")
     return 0, 0
```

```
raise IndexError(f"Index {index} is out of bounds for the
data (max index is {max row len-1})")
   for row in data:
      if index >= len(row):
           raise IndexError(f"Index {index} is out of bounds for
          value = float(row[index])
          values.append(value)
          raise ValueError(f"Non-numeric value found in column
index}: '{row[index]}'")
  return total, average
```

		total = sum(values) avg = total / len(values) return total, avg def main(): """" Main function to read the CSV, process data, and print summary. """" filepath = input("Enter the path to the CSV file: ").strip() data = read_csv(filepath) if not data: return # Exit if no data was loaded try: column_index = int(input("Enter the column index to summarize (starting from 0): ")) except ValueError: print("Error: Invalid column index.") return total, avg = summarize_column(data, column_index) if total != 0 or avg != 0: print("Total:", total) print("Average:", avg) else: print("No valid data to summarize.") ifname == 'main': main()	<pre>def main() -> None: """ Main function to execute the script. Reads data from 'data.csv', summarizes the second column (index 1), and prints the total and average. Handles potential errors during file processing and data summarization. """ filepath = 'data.csv' try: data = read_csv(filepath) total, average = summarize_column(data, 1) print(f"Total: {total}") print(f"Average: {average}") except (FileNotFoundError, IOError, TypeError, IndexError, ValueError) as e: print(f"Error: {e}") # Print the error message # Optionally, you could log the error or take other actions return # Exit the program more gracefully ifname == 'main': main()</pre>
Zero-shot	Analyze the following utility script. Identify any potential logic	import csv import os	# utils.py - Script to parse and summarize numeric CSV files import csv

issues or design flaws. Then, refactor it for better readability, correctness, and safety: # utils .py -

```
def read csv (
filepath ) :
with open (
filepath , 'r')
as f :
for row in csv .
reader (f)l
summarize column
for row in data
```

```
def read csv(filepath):
  Reads a CSV file and returns a list of rows.
Handles errors like file not found.
  if not os.path.exists(filepath):
     print(f"Error: The file at {filepath} does not
exist.")
     return []
  trv:
     with open(filepath, 'r') as f:
        return [row for row in csv.reader(f)]
  except csv.Error as e:
     print(f"Error reading the CSV file: {e}")
     return []
def summarize column(data, index):
  Summarizes a specific column (skipping
header). Returns the total and average of numeric
values.
  if len(data) <= 1:
     print("Error: The file contains no data to
summarize.")
     return 0.0
  values = []
  for row in data[1:]: # skip header row
     trv:
       value = float(row[index])
       values.append(value)
     except (ValueError, IndexError):
        print(f"Warning: Non-numeric or missing
value found at row {data.index(row)} in column
{index}. Skipping.")
       continue
  if not values:
     print("Warning: No valid numeric data found in
the column.")
     return 0.0
```

```
import os # For file existence check
from typing import List, Tuple, Optional # For type hinting
def read csv(filepath: str) -> List[List[str]]:
   if not os.path.exists(filepath):
       raise FileNotFoundError(f"File not found at {filepath}")
       with open(filepath, 'r', encoding='utf-8') as f: #
          reader = csv.reader(f)
           data: List[List[str]] = list(reader) # Read all rows
          return data
   except IOError as e:
       raise IOError(f"Error reading CSV file: {e}")
```

```
filepath = 'data
.csv '
  data = read_csv
( filepath )
  total , avg =
  summarize_column
( data , 1)
  print (" Total
:", total )
  print (" Average
:", avg )
if __name__ ==
'__main__ ':
  main ()
```

```
total = sum(values)
  avg = total / len(values)
  return total, avg
def main():
  Main function to read the CSV, process data,
and print summary.
  filepath = input("Enter the path to the CSV file:
").strip()
  data = read csv(filepath)
  if not data:
     return # Exit if no data was loaded
  try:
     column index = int(input("Enter the column
index to summarize (starting from 0): "))
  except ValueError:
     print("Error: Invalid column index.")
     return
  total, avg = summarize column(data,
column index)
  if total != 0 or avg != 0:
     print("Total:", total)
    print("Average:", avg)
     print("No valid data to summarize.")
if name == ' main ':
  main()
```

```
def summarize column(data: List[List[str]], index: int) ->
Tuple[float, float]:
       index: The index of the column to summarize (0-based).
      A tuple containing the total and average (float, float).
an int.
  if not isinstance(data, list) or not all(isinstance(row,
list) for row in data):
       raise TypeError("Input 'data' must be a list of lists.")
       raise TypeError("Input 'index' must be an integer.")
```

```
if index < 0 or any(index >= len(row) for row in data):
       raise IndexError(f"Index {index} is out of bounds for the
data.")
          value = float(row[index])
           values.append(value)
           raise ValueError(f"Non-numeric value in column
   total = sum(values)
   return total, average
def main() -> None:
```

```
Includes error handling.
  filepath = 'data.csv'
      data = read csv(filepath)
  except (FileNotFoundError, IOError, TypeError, IndexError,
ValueError) as e:
if name == ' main ':
```

BONUS Code Completion and Robustness Enhancement (Python)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Chain-of-thought s	Complete the following file-processing script. The goal is to clean each line,	Step-by-Step Refactoring of the file_processor.py Script The goal of this script is to clean text	<pre># file_processor.py - Script to process text files import string import os # For file existence check</pre>

remove punctuation, and count word frequencies correctly. Do it **step by step**:

```
import string
def load file
filepath ) :
filepath , 'r')
as f :
readlines ()
punctuation and
def count words
word counts =
lines :
```

by removing punctuation, converting text to lowercase, and counting word frequencies correctly. Here's a breakdown of the changes required to complete the script.

1. load_file() function:

 This function is mostly correct. It reads the lines from a file, so no changes are needed here.

2. clean_line() function:

 We need to complete the clean_line() function to remove punctuation and convert the text to lowercase.

• Plan:

- Remove punctuation: Use Python's string.punctua tion to identify punctuation characters.
- Convert to lowercase: Use the .lower() method to convert all characters in the line to lowercase.

3. count_words() function:

```
from typing import List, Dict # For type hinting
def load file(filepath: str) -> List[str]:
  if not os.path.exists(filepath):
       raise FileNotFoundError(f"File not found at: {filepath}")
      with open(filepath, 'r', encoding='utf-8') as f:
           lines: List[str] = f.readlines()
   except IOError as e:
       raise IOError(f"Error reading file: {filepath} - {e}")
def clean line(line: str) -> str:
```

```
clean line (
line )
 for word in
clean . split
  word counts
word counts .
get ( word , 0)
+ 1
word counts
def main () :
filepath =
load file (
filepath )
count words (
lines )
for word ,
count in sorted
items () ) :
  print ( f"{
word }: { count
```

- This function is mostly correct, but the code could be improved for clarity.
- We need to ensure the clean_line() function works correctly by removing punctuation and splitting the cleaned lines into words.

4. main() function:

 The main() function looks fine. It loads the file, processes the lines, counts the words, and prints the results in a sorted order.

Step 1: Complete the clean_line() function

We'll use Python's string.punctuation to remove punctuation and .lower() to convert text to lowercase.

import string

def clean_line(line):

Cleans the input line by removing punctuation and converting to lowercase.

Remove punctuation using str.translate() and string.punctuation translator = str.maketrans(", ",

```
A string with punctuation removed and converted to lowercase.
  translator = str.maketrans('', '', string.punctuation)
  cleaned line = line.translate(translator)
   cleaned line = cleaned line.lower()
  return cleaned line
def count words(lines: List[str]) -> Dict[str, int]:
  word counts: Dict[str, int] = {}
   for line in lines:
```

```
string.punctuation)
  clean =
line.translate(translator).lower()
  return clean
```

Step 2: Complete the count_words() function

We need to ensure the function correctly counts the words, and handle the cleaned line properly.

```
def count_words(lines):
```

Counts the frequency of each word in the provided lines.

word_counts = {}
for line in lines:
 clean = clean_line(line)
 for word in clean.split():
 word_counts[word] =
word_counts.get(word, 0) + 1
 return word_counts

Step 3: Final Script with All Components

Here's the complete refactored file_processor.py script:

import string

```
def load_file(filepath):
```

Reads the file and returns the lines as a list of strings.

with open(filepath, 'r') as f: return f.readlines()

```
cleaned line = clean line(line)
       for word in cleaned line.split():
   return word counts
def main() -> None:
   counts the word frequencies, and prints the results in sorted order.
   filepath = 'input.txt'
      lines = load file(filepath)
      counts = count words(lines)
       for word, count in sorted(counts.items()):
  except (FileNotFoundError, IOError) as e:
      print(f"Error: {e}") # Print the error message
if name == ' main ':
```

Step-by-step explanation:

- 1. Import necessary modules:
 - o string: Provides access to punctuation characters for removal.
 - os: Provides functions for interacting with the operating system, such as checking file existence.
 - o typing: Provides type hinting for function arguments and return values, improving code clarity

```
def clean line(line):
  Cleans the input line by removing
punctuation and converting to
lowercase.
  # Remove punctuation using
str.translate() and string.punctuation
  translator = str.maketrans(", ",
string.punctuation)
  clean =
line.translate(translator).lower()
  return clean
def count words(lines):
  Counts the frequency of each
word in the provided lines.
  word counts = {}
  for line in lines:
     clean = clean line(line)
     for word in clean.split():
       word counts[word] =
word counts.get(word, 0) + 1
  return word counts
def main():
  Main function to read file, process
lines, count words, and print the
results.
  filepath = 'input.txt' # Adjust the
file path as needed
  lines = load file(filepath)
  counts = count words(lines)
  # Print the word counts sorted by
word
  for word, count in
sorted(counts.items()):
     print(f"{word}: {count}")
```

and maintainability.

2. load_file(filepath: str) -> List[str]:

- Handles file loading with error handling:
 - Checks if the file exists using os.path.exists(filepath) and raises a FileNotFoundError if it doesn't.
 - Opens the file in read mode ('r') with encoding='utf-8' to handle various character encodings.
 - Uses a try...except block to catch potential IOError exceptions during file reading and raises an IOError with a more informative message.
 - Reads all lines from the file using f.readlines() and returns them as a list of strings.
- Includes a comprehensive docstring explaining the function's purpose, arguments, return value, and exceptions.

3. clean line(line: str) -> str:

- Implements the line cleaning logic:
 - Creates a translation table using str.maketrans(", ", string.punctuation). The first two
 arguments being empty strings and the third being the string of punctuation characters
 means that these characters will be deleted.
 - Removes punctuation from the input line using line.translate(translator).
 - Converts the cleaned line to lowercase using cleaned line.lower().
 - Returns the cleaned line.
- Includes a docstring.

4. count_words(lines: List[str]) -> Dict[str, int]:

- Counts word frequencies:
 - Initializes an empty dictionary word counts to store the counts.
 - Iterates through each line in the input lines.
 - Calls clean_line(line) to get the cleaned version of the line.
 - Splits the cleaned line into words using cleaned line.split().
 - For each word, it increments its count in the word_counts dictionary using word_counts.get(word, 0) + 1. This efficiently handles cases where a word is encountered for the first time.
 - Returns the word counts dictionary.
- Includes a docstring.

5. main() -> None:

- o Orchestrates the script execution:
 - Sets the filepath to 'input.txt'.

```
    Uses a try...except block to handle potential FileNotFoundError or IOError exceptions

                                        if name == ' main ':
                                                                                             during file loading.
                                          main()
                                                                                         ■ Calls load file(filepath) to get the lines from the file.
                                                                                             Calls count words(lines) to count the word frequencies.
                                                                                             Sorts the word counts alphabetically by word using sorted(counts.items()).
                                                                                         Prints the word counts in the format "{word}: {count}".
                                                                                        Includes a docstring and error handling.
Zero-shot
                  Complete the
                                        import string
                  following
                                                                              import string
                  file-processing
                                        def load file(filepath):
                  script. The goal is to
                                                                              import os
                  clean each line.
                                          Reads the file and returns the lines
                                                                              from typing import List, Dict
                  remove punctuation,
                                        as a list of strings.
                  and count word
                  frequencies
                                          with open(filepath, 'r') as f:
                                                                              def load file(filepath: str) -> List[str]:
                  correctly:
                                             return f.readlines()
                                        def clean line(line):
                                          Cleans the input line by removing
                                        punctuation and converting to
                                        lowercase.
                                          # Remove punctuation using
                                        str.translate() and string.punctuation
                                          translator = str.maketrans(", ",
                  import string
                                        string.punctuation)
                   def load file
                                          clean =
                                        line.translate(translator).lower()
                                          return clean
                   with open (
                  filepath , 'r')
                                        def count words(lines):
                  as f :
                                          Counts the frequency of each
                                        word in the provided lines.
                  readlines ()
                   def clean line
                                          word counts = {}
                                                                                 if not os.path.exists(filepath):
                                          for line in lines:
                                             clean = clean line(line) # Clean
                                                                                       raise FileNotFoundError(f"File not found at: {filepath}")
                                        the line by removing punctuation and
```

```
def count words
word counts =
lines :
clean line (
line )
for word in
clean . split
():
  word counts
word counts .
get ( word , 0)
word counts
def main () :
 filepath =
'input .txt '
filepath )
count words (
lines )
```

```
making it lowercase
     for word in clean.split(): # Split
the cleaned line into words
       word counts[word] =
word counts.get(word, 0) + 1 #
Count the words
  return word counts
def main():
  Main function to read file, process
lines, count words, and print the
results.
  filepath = 'input.txt' # Adjust the
file path as needed
  lines = load file(filepath)
  counts = count words(lines)
  # Print the word counts sorted by
word
  for word, count in
sorted(counts.items()):
     print(f"{word}: {count}")
if name == ' main ':
  main()
```

```
with open(filepath, 'r', encoding='utf-8') as f:
           lines: List[str] = f.readlines()
       raise IOError(f"Error reading file: {filepath} - {e}")
def clean line(line: str) -> str:
       A string with punctuation removed and converted to lowercase.
   translator = str.maketrans('', '', string.punctuation)
   cleaned line = line.translate(translator)
   return cleaned line.lower()
def count words(lines: List[str]) -> Dict[str, int]:
```

```
for word ,
count in sorted
( counts .
items () ) :
    print ( f"{
word }: { count
}")
if __name__ ==
'__main__ ':
main ()
```

```
cleaned line = clean line(line)
def main() -> None:
      lines = load file(filepath)
      counts = count words(lines)
      for word, count in sorted(counts.items()):
if name == ' main ':
```