

Using ChatGPT as a Coding Assistant

Author: Mark Baltič, Arctur

ChatGPT has become a valuable tool in software development, capable of generating code quickly and providing useful insights. However, directly copying large code snippets into a project without review can introduce unintended bugs, often leading to time-consuming debugging. In some cases, manually writing code from scratch may be more efficient.

Despite these challenges, there are several scenarios where ChatGPT adds significant value to coding workflows. Below are some practical examples, focusing on Python, though the same concepts can be applied to other programming languages. The goal is not just to solve problems but to demonstrate how to use ChatGPT effectively as a development assistant.

Assigning a Role for Specialized Insights

ChatGPT can be a powerful tool when treated as a specialized assistant rather than a general-purpose chatbot. By assigning it a role, it is possible to gain expert-level insight into various aspects of software development. For instance, ChatGPT can be asked to act as a senior software engineer, DevOps specialist, or database architect. This approach provides more tailored advice and helps streamline the development process.

Example Use Case: Optimizing a Data Aggregation Function in Python

When developing a Python application that processes user data and calculates statistics based on user ages, it is often necessary to optimize performance, especially when working with large datasets. The following example demonstrates how ChatGPT can assist in improving the efficiency of such functions.

```
def process_user_data(users, age_threshold):  
    """Counts users above a certain age and calculates their average age."""  
    count = 0  
    total_age = 0  
  
    for user in users:  
        age = user.get("age")  
        if age and age > age_threshold:  
            count += 1  
            total_age += age  
  
    if count == 0:  
        return 0, 0  
  
    average_age = total_age / count  
    return count, average_age
```

A function that counts the number of users above a certain age threshold and calculates their average age may initially be implemented in a straightforward manner. However, performance concerns may arise when processing large amounts of data.

A structured prompt can be used to assign this optimization task to ChatGPT:

Example Prompt:

"You are a performance optimization expert specializing in data processing with Python. Analyze the following function and suggest ways to improve its execution speed for large datasets. Focus on computational efficiency and clean code practices. Provide code examples if possible."

ChatGPT provides multiple suggestions for optimizing efficiency. In this case, the response includes the following recommendations:

- **Avoid Redundant Loop Operations:** Suggest combining the counting and sum calculation in a single step more efficiently.
- **Leverage Functional Programming:** Use tools like `filter()` and `sum()` to simplify and speed up computations.
- **Consider External Libraries:** Recommend using `pandas` for better performance when processing large datasets.

To explore the second suggestion further, an additional request can be made for an optimized implementation and performance testing.

Example Follow-up Prompt:

"Provide the code for the second suggestion and add code for testing the performance of the initial code and the optimization."

ChatGPT returns the optimized function along with benchmarking code to compare execution times.

```
def process_user_data_functional(users, age_threshold):
    """Optimized implementation using sum and list comprehensions."""
    valid_ages = [user["age"] for user in users if user.get("age", 0) > age_threshold]

    count = len(valid_ages)
    total_age = sum(valid_ages)

    if count == 0:
        return 0, 0

    return count, total_age / count
```

```
import time
import numpy as np
def measure_performance(func, users, age_threshold):
    """Measures the performance of a given function."""
    start_time = time.time()
    result = func(users, age_threshold)
    end_time = time.time()
    execution_time = end_time - start_time

    print(f"{func.__name__}: {execution_time:.6f} seconds")
    print(f"Total Users Above Age {age_threshold}: {result[0]}, Average Age: {result[1]:.2f}\n")
    return execution_time

# Generate a large dataset of user ages
users_data = [{"name": f"User{i}", "age": np.random.randint(18, 70)} for i in range(10_000_000)]

# Measure and compare performance
initial_time = measure_performance(process_user_data, users_data, 30)
functional_time = measure_performance(process_user_data_functional, users_data, 30)

print("Performance Summary:")
print(f"Initial Code (Loop): {initial_time:.6f} seconds")
print(f"Functional Programming (sum): {functional_time:.6f} seconds")
```

Running the provided test confirms an improvement in efficiency.

```
process_user_data: 1.716974 seconds
Total Users Above Age 30: 15000090, Average Age: 50.00

process_user_data_functional: 1.653179 seconds
Total Users Above Age 30: 15000090, Average Age: 50.00

Performance Summary:
Initial Code (Loop): 1.716974 seconds
Functional Programming (sum): 1.653179 seconds
```

Generating Documentation

ChatGPT is highly effective in analyzing code functionality and generating appropriate documentation to improve code clarity.

Example Use Case: Generating Function Documentation for Python Code

Once a function has been implemented, it is useful to include documentation to describe its purpose and usage. A well-crafted prompt can help automate this process.

```
def process_transactions(transactions, min_amount):  
    """Filters transactions above a minimum amount and calculates total value."""  
    valid_transactions = [t for t in transactions if t["amount"] >= min_amount]  
    total_value = sum(t["amount"] for t in valid_transactions)  
    return valid_transactions, total_value
```

Example Prompt:

"You are an expert technical writer. Write a professional Python docstring for the function above."

Chat GPT returns the function documentation according to the desired standard:

```
def process_transactions(transactions, min_amount):  
    """  
    Filters transactions that meet or exceed a specified minimum amount  
    and calculates the total value of valid transactions.  
  
    Parameters:  
        transactions (list of dict): A list of transactions, where each  
        transaction is a dictionary containing at least a key `amount`.  
        min_amount (float): The minimum transaction amount to include in the total.  
  
    Returns:  
        tuple: A tuple containing:  
        - valid_transactions (list of dict): Filtered transactions meeting the minimum amount.  
        - total_value (float): The sum of valid transaction amounts.  
  
    Example:  
        >>> transactions = [{"id": 1, "amount": 100}, {"id": 2, "amount": 50}]  
        >>> process_transactions(transactions, 60)  
        ([{"id": 1, "amount": 100}], 100.0)  
    """  
    valid_transactions = [t for t in transactions if t["amount"] >= min_amount]  
    total_value = sum(t["amount"] for t in valid_transactions)  
    return valid_transactions, total_value
```

Keep in mind, that ChatGPT can provide you documentation in different formats! Here is a documentation for the same function using Sphinx Format:

```
def process_transactions(transactions, min_amount):
    """
    Filters transactions that meet or exceed a specified minimum amount
    and calculates their total value.

    :param transactions: A list of transactions, where each transaction is
        a dictionary containing at least a key `amount`.
    :type transactions: list[dict]
    :param min_amount: The minimum transaction amount to include in the total.
    :type min_amount: float

    :return: A tuple containing:
        - list of valid transactions that meet the minimum amount.
        - The sum of all valid transaction amounts.
    :rtype: tuple[list[dict], float]

    :Example:

    .. code-block:: python

        transactions = [
            {"id": 1, "amount": 100},
            {"id": 2, "amount": 50},
            {"id": 3, "amount": 200}
        ]
        min_amount = 100
        result = process_transactions(transactions, min_amount)
        print(result) # ( [{"id": 1, "amount": 100}, {"id": 3, "amount": 200}], 300.0)
    """
    valid_transactions = [t for t in transactions if t["amount"] >= min_amount]
    total_value = sum(t["amount"] for t in valid_transactions)
    return valid_transactions, total_value
```

Writing test cases

Writing test cases is an essential part of software development, ensuring that code behaves as expected. ChatGPT can assist by **generating structured unit tests**, **identifying edge cases**, and **providing best practices**. Keep in mind that double-checking the provided test cases should become a habit.

Example: Writing Unit Tests for process_transactions Function

Using the previously introduced function, unit tests can be generated with a targeted prompt:

Example Prompt:

"Write unit tests for the function process_transactions using Python's unit test module. Cover normal cases, edge cases, and invalid input scenarios."

The output consists of suitable test cases for our function:

```
class TestProcessTransactions(unittest.TestCase):
    """Unit tests for the process_transactions function."""

    def test_normal_case(self):
        """Test with a standard set of transactions."""
        transactions = [
            {"id": 1, "amount": 100},
            {"id": 2, "amount": 50},
            {"id": 3, "amount": 200}
        ]
        min_amount = 100
        expected_result = ([{"id": 1, "amount": 100}, {"id": 3, "amount": 200}], 300.0)
        self.assertEqual(process_transactions(transactions, min_amount), expected_result)

    def test_all_transactions_valid(self):
        """Test when all transactions exceed the minimum amount."""
        transactions = [
            {"id": 1, "amount": 500},
            {"id": 2, "amount": 700},
            {"id": 3, "amount": 1000}
        ]
        min_amount = 100
        expected_result = (transactions, 2200.0)
        self.assertEqual(process_transactions(transactions, min_amount), expected_result)

    def test_no_transactions_valid(self):
        """Test when no transactions meet the minimum amount."""
        transactions = [
            {"id": 1, "amount": 10},
            {"id": 2, "amount": 20},
            {"id": 3, "amount": 30}
        ]
        min_amount = 100
        expected_result = ([], 0.0)
        self.assertEqual(process_transactions(transactions, min_amount), expected_result)
```

As well as edge and special cases:

```
def test_empty_transaction_list(self):
    """Test when the transaction list is empty."""
    transactions = []
    min_amount = 100
    expected_result = ([], 0.0)
    self.assertEqual(process_transactions(transactions, min_amount), expected_result)

def test_edge_case_exact_threshold(self):
    """Test when some transactions are exactly at the threshold."""
    transactions = [
        {"id": 1, "amount": 100},
        {"id": 2, "amount": 200},
        {"id": 3, "amount": 100}
    ]
    min_amount = 100
    expected_result = ([{"id": 1, "amount": 100}, {"id": 2, "amount": 200}, {"id": 3, "amount": 100}], 400.0)
    self.assertEqual(process_transactions(transactions, min_amount), expected_result)

def test_invalid_input(self):
    """Test handling of invalid input (non-numeric amounts)."""
    transactions = [
        {"id": 1, "amount": "100"}, # Invalid: String instead of number
        {"id": 2, "amount": None}, # Invalid: None type
        {"id": 3, "amount": 200}
    ]
    min_amount = 100

    with self.assertRaises(TypeError):
        process_transactions(transactions, min_amount)
```

This provides a great baseline for building the final version of test cases.

Conclusion (written by chatGPT based on the text above)

ChatGPT streamlines software development by assisting with specialized insights, documentation and test case generation. While it doesn't replace human expertise, it enhances productivity and efficiency, making it a valuable tool for developers.