**ChatGPT**

Supported languages: ChatGPT provides general coding help and advice in various programming languages. Primarily designed for natural language interaction, ChatGPT (also known as Microsoft Copilot) can understand and generate answers in several human languages. It also has the ability to understand programming concepts and code snippets in several programming languages, although this is not its primary function. It is also worth noting that Microsoft Copilot is free for students and is built on the GPT4 version, so instead of the paid version, you can use the most powerful version of ChatGPT for free with Microsoft 365 services integration.

Specifications: ChatGPT is great for generating explanations, writing code snippets, and answering theoretical questions.

ChatGPT uses a variant of the Generative Pre-trained Transformer (GPT) model, which is designed for a wide range of text-based tasks. It offers extensive coding support and suggestions, making it a versatile tool for developers looking for general guidance.

**GitHub Copilot**

Supported Languages: Provides extensive support for major programming languages, including JavaScript, Python, TypeScript, Java, C++, and C. It is based on the OpenAI Codex model. GitHub Copilot supports a wide range of programming languages, making it versatile for developers working in different environments.

Specifications: Microsoft Copilot acts as a rapid coding assistant, providing real-time code completion, snippet generation, and documentation. GitHub Copilot acts as an integrated AI pair programmer for Visual Studio Code, providing complete lines or blocks of code based on comments and code context, using machine learning models trained on a large code base. It dynamically adapts to the coding style and requirements of the user's project. It is particularly strong in code writing and completion, suggestions, and understanding the context of the code being written.

**BlackBox**

Supported languages: BlackBox has been noted to support a wide range of programming languages, is known for its specialization in specific programming languages ​​and writing code with explanations.

Specifications: BlackBox is useful for more in-depth code analysis, performance optimization recommendations, and debugging complex algorithms. It has a Playground feature. There is no chat collection/history feature, but it is possible to open two or more chats in one window at the same time. In addition, it is also possible to create various personal chatbot assistants loaded with specific knowledge online or download an extension application to the VS Code IDE (See Figure 3. Main integration options. BlackBox. “Agents” and “playground”).

BlackBox features include real-time collaboration, error detection and program performance analysis from images or code, code optimization tools, and possible AI-based code recommendations

**How to use AI tools**

**First exercise method. Code Assistant**

How to solve given tasks

\* Break down your coding goal into smaller subtasks (coding tasks). Then, mentally design how you would guide another developer to tackle each step. This will help you provide a logical sequence of AI-focused queries.

\* Ask about parts of the AI ​​theory or task that are too complex to understand, and also ask for answers with examples. Use AI as an idea generator and integrate ideas or examples, commenting and rewriting the code according to the conditions and requirements.

\* For complex coding tasks, break the work down further into . Ask about each stage separately.

1. Describe the different situations that a program might encounter and how to handle these situations.

2. Have the AI ​​produce clear documentation and comments that explain each part of the code. Ask for a short explanation, step by step, with the purpose of the code in the comments. Well-documented code makes it easy for future editors to understand the AI’s logic and intentions.

3. Test the AI ​​code thoroughly and check it for bugs, performance issues, semantic errors, and security holes.

However, at the end of the practical assignment, the instructor should review the graphically constructed or generated algorithms, or conduct a questionnaire/discussion to understand the level of understanding the student has had of the theory and whether he or she can practically handle the homework.

This approach can be used for any practical assignment when introducing new theoretical material, but the author recommends setting aside a few hours two weeks before the first homework assignment to check that all students are participating at about the same pace. If someone falls behind, this method allows them to catch up and complete the assignment on time.

**Exercise 2: Code Review**

1. Provide 2-3 examples that show code that you or someone else has created.

2. Briefly explain the purpose of the code. Ask for suggestions on what might be wrong and how to fix it.

3. Provide examples of edge cases, errors (error messages), and boundary conditions that you want the logic to handle.

4. Ask the AI ​​to provide clear comments and analysis that explain each part of the code, along with suggestions for improvements.

5. Use the knowledge provided to modify the code.

**Exercise 3: Refactoring**

Students must choose a topic and one activity on that topic, ranging from sorting techniques to memory allocation.

Steps in the workflow:

1. Formulate the main tasks and questions about the topic.

2. Provide 2-3 examples that show the kind of code you want to create. It is important to mention the different extracts that format the data.

3. Use the "Rebuild" function Rebuild will present a new version, as it is possible to view all versions for a given code. However, BlackBox does not have a version saving function, the resulting code or response can be saved to a document for later comparison. Or, it is possible to open a new conversation in the same window in the web version

4. Review the AI ​​results and highlight the coding differences. Consider factors such as efficiency, readability, and modern coding practices.

Viewing and analyzing examples of different scenarios will increase your knowledge base, which in its own way will allow you to choose the optimal solution for each specific task.

**Fourth exercise type. Collaboration**

Students should be divided into pairs so that each person does 2 exercises. The exercises consist of 2 parts.

The first part of the exercise is the implementation of a given function by creating a correlation diagram that can be translated into pseudocode using AI.

The second part of the task involves implementing the diagram/pseudocode obtained from the first part of the work into an algorithm written in a programming language (for example, С), again using AI.

Necessary steps for the work:

1. Take different exercises.

2. Familiarize yourself with the function.

3. Create a correlation diagram (UML algorithm) that includes the objects, subfunctions, classes, etc. necessary for the exercises.

4. Contact the AI ​​to get pseudocode.

5. Check the correctness of the resulting code.

6. Repeat step 4 if necessary.

7. Exchange the results with your partner and your teammate's code.

8. Use the AI's tips and write/ask the AI ​​to write working program code in a programming language.

In conclusion, the group work was not only a technical exercise, but also an important tool for developing social and professional skills.