Mastering Coding Assistants With CoPilot:

[Visual Studio Edition]  
**Exercises / Tutorial**

**Setting Up A new Project**

1. If you don’t have a project yet and you are using (a language such as C#) create a new project where you can run simple console applications.  
   If you are using C#:  
   \* install the VS Code C# extension ( lighter, enough for exercises )  
    or the VS Code Dev-Kit extension ( heavier, requires microsfot login )  
   \* create a new folder  
   \* in VS code choose open folder  
   \* in terminal, in that folder type ‘dotnet new console’  
   \* accept the msg box to add required assets ( bin,obj dirs etc ), or type ‘code .’ if it doesn’t come up

**Code Completions Exercises**

1. **Using Code Suggestions**  
   Code Suggestion Only - Do not use chat for this exercise  
   In [program.cs](http://program.cs) start typing ClassSquareRootCalculator  
   As the code comes up in ghost text select tab to accept it  
   Save yoru code  
   Run your code using dotnet run or debug  
   Verify that it works correctly to calculate the square root of a number
2. **Accepting Partial Suggestions**  
   In your class, begin typing PrintTheSquareRootsOf1To10  
   Press Ctrl + rightArrow to accept the suggested completion one word at a time  
   Accept the entire function and call it from main to see that it works.
3. **Cycling Through Suggestions**  
   Above the function name begin typing a comment  
   // to use  
   Try to cycle through the AI suggestions by pressing Ctrl + ] or hovering with the mouse  
   If it doesn’t offer more than one, try changing the text  
   Note: C# keyboard conflict may cause text tol also tab-right while cycling suggestions  
   Press tab to accept one of the suggestions
4. **Disabling Suggestions**  
   Erase a few lines from the function, and then start typing it again.  
   Notice that the ghost text is back.  
   Now disable code completions on the CoPilot icon on the bottom right.  
   Start typing again and see that there are no completions  
   You may now re-
5. **Generating Code By Writing Comments**Using **only comments**, get the AI Completion to generate a class called StringPermutator which has two functions, RandomPermuation and AllPermutations which create permutations of strings.  
   RandomPermutation returns one permutation String ( e.g. abc -> cba )  
   AllPermutations returns a list of all possible permutations ( e.g. abc -> [abc acb bac bca cab cba ] )  
   Using another comment, create code that will test your functions and run it
6. **Next Edit Suggestion (NES)**  
   Make sure NES is enabled (easiest using copilot icon on lower right )  
   Create a class called Student with a constructor that takes a firstName and lastName parameters and sets them into member variables/properties.  
   Edit the class to add a third member/property called MiddleName  
   Access the NES suggestions for modifying the constructor accordingly  
     
     
   **Inline Chat Exercises**
7. **Using Inline Chat**  
   Clear all code from the editor or create a new file to run  
   Click - open-chat or use Ctrl+I to open the in editor chat  
   **For this exercise use Inline Chat only. Do not use the chat window**  
   This should bring a chat window right in the current editor   
   Get the AI to create a basic calculator that accepts strings from the user and can handle + - \* / on two operands  
   Sample inputs 5 + 3 outputs 5 + 3 = 8  
    2 - 1 2 - 1 = 1  
    3 \* 5 3 \* 5 = 15
8. **Making Code Changes With Inline Chat  
   Remember: For this exercise use Inline Chat only. Do not use the chat window**  
   In the same exercise , place the cursor before the line where the user input is read.  
   Tell the AI to create a beautiful text splash screen before each time the input is read
9. **Remember: For this exercise use Inline Chat only. Do not use the chat window**Select all the code, and tell the AI to refactor it into smaller functions. Did it work?  
   Undo your changes.  
   Place the cursor at the top of the class.  
   Tell the AI to refactor it into smaller functions. Did it work?  
   Undo your changes.  
   Place the cursor at the middle of the class.  
   Tell the AI to refactor it into smaller functions. Did it work?
10. Bonus: Using only the inline editor, make the calculator able to ignore white spaces, support multiple operands, and support operator precedence  
    Input: 5+ 9\*2 output: 5 + 9 \* 2 = 23
11. **Changing the suggested code**  
    Erase all the code in the editor.  
    With the Inline-Chat ask the AI to create a main that puts some hard coded numbers in a list, and calls a function to sort the list, and then prints the numbers in the list.  
    **Do not accept or cancel the suggested answer yet**Instead of accepting the suggested answer, tell the AI you want to make a change in the proposed code. For example, tell it to leave the original list untouched, and return a new sorted list instead.  
    While in the chat , try using the up and down keys to see your previous commands
12. **Terminal Inline Chat**Place the cursor in the terminal window.  
    Press Ctrl+I and tell the AI to show you how to run your code  
    If using C#, ask the AI how you can clean and remove your bin and obj directories  
    Paste the suggested command into the terminal using the dedicated ‘Insert-Into-Ternubak’UI button  
    Use the up and down arrows to see your previous AI commands.  
      
      
    **Using The Chat Window**
13. **Using Ask Mode**Clear any code you have in the file or create a new file  
    Press Ctrl+Alt+I or open the chat window from the UI  
    Make sure you are in **ASK** mode  
    Ask the AI a question such as “what is the template method pattern?”  
    Tell the AI to give you a full running example of what you asked about. For example “give me a working example of the template method pattern so I can see it run”  
    Insert the given code into your editor using one of the dedicated UI buttons “Apply to program” or “Insert at cursor”  
    Run and make sure it works.  
    Ask the AI how many functions you have. Check if it’s correct.
14. **Using Edit Mode**Erase all your code.  
    Start a new AI session in **Edit Mode**Tell the AI to provide you with a working example of the same thing you did in question 14.  
    Keep your changes and run the code.
15. **Refactoring with Edit Mode**Tell the AI to move Main to the top of the file ( or bottom if it’s at the top already )  
    Keep your changes and run.  
    Tell the AI to move every class into it’s own file.  
    Keep your changes and run again.  
    Look at each file to see what the AI put in it.
16. **Using Chat Undo**Use the undo icon at the top of the chat window to undo your changes one at a time.  
    Check the program structure after every one.
17. **Fixing Errors With Edit Mode**Put a deliberate error in main ( for example type gibberish like asdfadfdf )  
    Run the program from the terminal so you get errors.  
    Tell the AI that you are getting run errors and watch it fix it.
18. **Missing Context With Edit Mode**???
19. **Working With Agent Mode**Tell the AI to create a text based 1 player vs computer tic tac toe game where the computer tries to win using the min-max algorithm and that it should make the program cleanly coded with nice small classes, each in its own file.  
    Watch the game run .
20. **Agent Mode Vs Edit Mode**Ask **Agent Mode** to build or re-build your code for you.   
    Then, try **Edit mode** the same thing.  
    In **Agent Mode**, type an incorrect build command such as dotnet buil  
    Ask the Agent Mode chat what went wrong with your build.  
    Then, try the same thing in a new **Edit Mode** session.
21. **Use Vision**Use paper or a paint program to draw an emoji such as a smiley face.  
    Ask the AI to create a program that outputs a text graphics version of it.
22. **Undoing History and Rolling Back Work**Starting with no code.Tell the AI to create a Simple calculator with Add and Subtract Functions  
    Now tell the AI to add a Multiplication function  
    Now tell the AI to add a subtraction function  
    In the chat history, go back to the “add Multiplication” instruction and delete it.  
    Make sure that only the add and subtract functions remain.  
    Try the same thing with the “Undo” icon.
23. **Use Chat Commands**Tell the AI to build a guess the number game ( computer chooses number 1 to 100,user guesses, computer answers higher/lower, user gets 10 tries )   
    Run it and see that it works.  
    Put a deliberate error in the code ( e.g. make the random between 1 and 1 )  
    In the chat , type /fix wrong numbers  
    Make sure the AI fixed the code.  
    In the chat type /save to move your chat to an editor window  
    In the chat type /clear to clear your chat
24. Using @Mentions  
    In a new chat window, ask the AI ‘how do I create a new file?’  
    Now compare that with ‘@vscode how do I create a new file?’
25. Using #Mentions  
    Create a program with a warning.  
    For example, if you are using c# and SDK 9, this will work  
    ;

using System  
class Program

{

static void Main(string[] args)

{

string input = Console.ReadLine();

Console.WriteLine(input);

}

}

In a new **Edit Mode** chat window compare the following two questions:  
 what problems do I have?  
 what #problems do I have?  
**Agent Mode** may behave differently.  
  
  
**Smart Actions**

1. **Getting AI Name Suggestions**  
   Write some code in a function that declares a local variable.  
   For example, you can use the Console.ReadLine code from the previous exercise.  
   Use F2 to rename the variable. Wait for coPilot to suggest some names.
2. **Generating Docs**  
   Put the cursor on the line declaring the function.  
   Right click and select CoPilot - generate docs - copilot will generate a function description.  
   Before accepting , in the inline chat ( where it says ask copilot ) type “make it short”. CoPilot should shorten the function description.  
   Press close in the inline-chat to reject CoPilot’s suggestion.  
   Press Ctrl-I and in the inline-chat window that comes type /doc [some text] for example /doc Main   
   /doc short explanation  
   /doc with emojis
3. **Sparkly Smart Actions**  
   Create an uninitialized local variable, for example, if using c#  
   int i;  
   Place the cursor on the line declaring the variable and wait for the sparkly icon or light bulb with sparkly, and then click it.  
   Choose ‘explain using copilot’ and read the warning’s explanation.  
   Choose ‘fix using copilot’  
   If using C#, get copilot to fix the warning on the line   
   string input = Console.ReadLine();
4. **Creating Unit Tests**  
   Ask CoPilot to create a FactorialCalc class  
   FactorialCalc.CalcFactorial(5)=120  
   Accept the proposed class.  
   Right click the class name and choose CoPilot->Generate Tests  
   Accept the proposed code and save it.  
   **\* If the tests don’t compile or there are build issues, don’t worry about that right now. We’ll get to that exercises later.**
5. **Getting Code Reviews**  
   **Temporarily Turn Off Code Completions!**Without using AI help, write, by yourself, a function to find the second largest integer in an array

static void Main(string[] args){

Console.WriteLine(FindSecondLargest  
 (new int[] { 3, -1, 2, 9, -5, 100, 22, 3 }));

}

static int FindSecondLargest(int[] data)

After your code compiles, place the cursor on your function and right click copilot->review and comment. If your review does not produce anything, add an obvious bug and try again.  
Compare this to switching to the chat window and asking it to review your function.  
  
  
**Longer Exercises**

1. **Setting Up A Test Solution**  
   With CoPilot’s help, create a simple project called FactorialCalc with an NUnit Test Subproject.  
   Make sure that everything builds, and the tests are discovered and run correctly.
2. **Working With Agents**  
   Using Visual Studio code and Copilot, tell the assistant to create a program that creates 10 number literals, puts them in a list and sorts them using bubble sort. Ask the LLM to organize everything nicely and cleanly in functions.  
   Part 2:  
   Ask the LLM to separate the different functions into different files for you.
3. **Creating A Graphics Game**Using Visual Studio code and Copilot or another coding assistant ask the LLM to create a 2 player graphics based tic-tac-toe game.  
   The board should have a **Graphical UI** and will look something similar to   
    x | |  
    —----------  
    | O |  
    —----------  
    | |  
     
   Each player will input their next move by using the mouse.  
   The game will determine when one player has won.
4. Change the game so that it is one player vs the computer. Recommended algorithm: Min Max
5. **Video Processing**Using Visual Studio code and Copilot or another coding assistant ask the LLM to create a program [python is recommended] that will reverse an mp4 video. Using open source 3rd part libraries is recommended.
6. **Video Game**Ask Visual studio ( or VS Code or Windsurf or cursor ) to create a video game for you.  
   Work in agent mode.  
   An example game is one where you have to click on objects appearing on the screen to get points.  
   If errors occur, ask the agent to run the game, look at the errors and do what is needed to fix them, including missing installations if any.  
   Part 2:  
   Ask the agent to improve your game and make it more enjoyable and likely to be played.
7. **Advanced Video Game**Ask cursor IDE to create a node based space invaders game for you, with graphics and sounds.
8. **MCP**  
   Use Visual Studio Code in Agent mode to call an MCP server:  
   In a visual studio workspace, under the .vscode folder create an mpc.json file  
   Add a server entry for an MPC server called demo-server1  
   Create an mpc server using docker or a local installation form the mpc server demos  
   <https://github.com/modelcontextprotocol/servers>  
   (For example you can use the Time sever)  
   Ask Visual Studio agent mode to call the MPC Server for you
9. **Custom MCP**  
   Optional: Write your own MCP Server  
   Follow a demo tutorial to create your own MCP server  
   For example, if using python you can follow the instructions under  
   <https://github.com/modelcontextprotocol/python-sdk>  
   Add a tool called “time\_nanoseconds” to your custom MPC server which returns the current system time in nanoseconds .  
   Ask the AI to create some functions that print various text messages, and have each function name suffixed by the current time nanoseconds as part of the function name.
10. **LLama Local LLM**Optional: Install and download Ollama to your machine  
    Choose one of the models and download it locally.  
    Run ollama in chat mode and interact with the LLM.   
    Turn off your internet connection and check that the LLM is still answering  
    In CoPilot , switch the Chat Window to use Ollama and see that you are getting responses even without an internet connection.
11. **Working With Your Own Code**Go to your own code from work, and continue a difficult task while applying what you’ve learned in this course.
12. Need more ideas for Vibe coding for today?  
    Create a photoshop style program with various filter effects such as blur, inverse image

Create a video editor style program with effects such as cut paste, reverse video

Create a stock price predictor based on historical price data.

Create a stock sentiment analyzer.

<https://github.com/rob-bar/vibe-code-apps-start/tree/main/1-menu-card>

<https://zapier.com/blog/vibe-coding-examples/>

<https://www.nucamp.co/blog/vibe-coding-top-10-vibe-coding-projects-that-can-make-you-real-money>