AI-Guided Practice in OCaml and Grammars

Use **ChatGPT, Claude, Gemini, or Grok** as a practice tutor. Give the AI the in-class examples in this assignment, ask it to create new exercises at a similar level, solve the exercises yourself, and then ask for feedback.

Part	Topic	Exercises
1	OCaml function types	5
2	Writing OCaml functions	6
3	CFG, BNF, and EBNF	6

Required Workflow

1. **Provide the examples.** Copy the relevant in-class examples into your chat. Ask for new exercises that practice the same ideas at about the same difficulty.
2. **Request exercises without answers.** Tell the AI: **"Do not give me the answers yet."**
3. **Make your own attempt.** Solve each exercise yourself, then send your answer and reasoning or code to the AI.
4. **Ask for feedback and revise.** Have the AI check your work and explain any mistakes. Make your own corrections, then ask it to check your revision.
5. **Keep the full conversation.** Preserve the generated exercises, your initial attempts, the feedback, and your revisions for submission to D2L.

Suggested Prompt

> *"I am practicing for CMSC 330 Programming Languages. Below are examples we discussed in class. Use them as examples of the topics and difficulty. Create new exercises for me, but do not give me the answers. I will solve each exercise first. After I send my answer, check it, explain any mistakes, and let me revise it."*

What to Submit to D2L

Submit the **complete, readable AI chat history for Parts 1-3**. Include the in-class examples you gave the AI, all generated exercises, your first attempt at each exercise, the AI feedback, and any corrected or revised work. If you use more than one tool or conversation, include every relevant chat.

Your own attempt must appear before the AI provides feedback or solutions.

Part 1. OCaml Function Types

Practice inferring types for recursive, curried, and higher-order OCaml functions.

IN-CLASS EXAMPLE 1

```
let rec accumulate f x lst =
    match lst with
    | [] -> x
    | y :: ys -> accumulate f (f x y) ys
```

Questions discussed in class

1. What is the type of `f`? **Note that** `x` **and** `y` **may not have the same type.**

2. What is the type of `accumulate`?

3. What does `accumulate` do?

Your task

Give this example to the AI and request **5 new type-inference exercises**. The set should include recursion, lists, currying, and higher-order functions. At least two exercises should require careful reasoning about type variables, including cases where arguments do not have to share the same type.

For each exercise, first determine the function's type yourself. Explain how the function body constrains its type and briefly describe what the function does. Then ask the AI to check your work.

Part 2. Writing OCaml Functions

Ask the AI to create **6 new OCaml programming exercises** to solve from scratch. Focus on basic function writing, recursion, currying, and tail recursion. **Do not use tuples yet.**

Exercises	Required focus
2	Basic recursive functions
2	Curried functions
2	Tail-recursive functions using a helper function and an accumulator

The exercises may use integers, floats, booleans, strings, and lists. They should not rely on tuples, records, objects, references, or advanced library functions that bypass the intended recursion practice.

Write each function yourself before sending it to the AI. Ask for feedback on correctness, recursion, and currying. When tail recursion is required, ask the AI to check whether the function is actually tail recursive.

Part 3. CFG, BNF, and EBNF

Practice designing and analyzing programming-language grammars, including precedence, associativity, BNF, EBNF, and ambiguity. Give the AI both in-class examples below.

IN-CLASS EXAMPLE 2 | EXPRESSION GRAMMAR

Write **unambiguous rules** for expressions with +, `/`, and `()`. Precedence is `()`, then `/`, then +. Parentheses have the highest precedence.

Add a rule to our `<expr>` grammar for exponentiation `**`, with precedence one level higher than division `/`. This operator should be **right associative**.

IN-CLASS EXAMPLE 3 | BNF TO EBNF

```
<expr> -> <expr> + <term>
        | <expr> - <term>
        | <term>

<term> -> <term> * <factor>
        | <term> / <factor>
        | <factor>
```

In-class question: What does the EBNF look like?

Your task

Use the two examples above to request **6 new grammar exercises**. The set must cover all of the following:

1. Design a CFG for a small piece of programming-language syntax.
2. Write an unambiguous expression grammar with specified operator precedence.
3. Add an operator to an existing expression grammar with a specified precedence and associativity.
4. Convert BNF to EBNF.
5. Determine whether a given CFG is ambiguous.
6. Show that a CFG is ambiguous by giving two distinct parse trees for the same string, or two distinct leftmost derivations (or two distinct rightmost derivations) of that string.

Attempt each exercise before asking the AI to check it. Explain how your grammar enforces the requested precedence and associativity, or why it is ambiguous. Review the feedback carefully rather than accepting it automatically.