1. I did task 1. My partner and I came up with different ways for task 3, after comparing our thoughts, my partner's way was more approachable, so we pair coded it together.

2. For typeo function, we checked to see if the input expr is a constant (numbero, stringo, and boolo) and set the type to the appropriate type. Second, we check if the expr is a symbolo but not a builtin symbol (not '+, '-, '*…) with the helper function not-builtino, if it satisfies both conditions, we know it's an identifier and we call lookupo to lookup the identifier in the env. Third, if the expr is a symbolo ('+, '-, '*…) we call the helper function builtino and deal with each symbolo differently. Fourth, for function calls, we call the helper function type-listo and get a list of types, then we check if the builtin arguments equal with list of types or else return 'error. Lastly, for function definitons, we check that the expr starts with a 'lambda and then we basically separate to two conditions where the first condition is the body of the expr is a pair and when it is not a pair. If it is a pair, we call helpero to get a list of argument types for the expr body. If not a pair, we create a new env and fill the ids with its types then for the body we check if it's a numbero, stringo, boolo, symbolo and set their types to their according types.
When handling function definitions for typeo it is different from typeof because we need to create new envs and fill it with ids, for example, (lambda (x y) 3), even though we are not using x and y we still need to evaluate it to ((? ?) 'num) where we don't care about the '? Therefore, we fill it up with ((_.0 _.1) 'num)

3. An advantage for using an interpreter is we can be more detailed with the instructions we are building, such as scoping or evaluation (we used many instructions in order to define simple operations). Macros is considered as a higher-level abstraction that allows us to extend an existing languages syntax to create more complicated syntax, but since it is abstract, we cannot look for details.