**Question 4.1a**
OCaml requires declaration of a variable before use unless passed the variable is passed into a function. In this case, the type can be inferred and the variable need not be re-declared. Variables are usually declared with the “let” syntax, e.g. let x = 4;; would be valid where simply saying x=4 without the “let” is invalid.

**Question 4.1b**
OCaml does not support operator overloading. The language was intentionally implemented this way to encourage the use of modules and functors instead and to maintain clarity of code.

**Question 4.1c**
Most of OCaml’s bindings are done at compile time, though inferred variables use runtime binding. OCaml uses a generic syntax of ‘a ‘ to signify that a generic object can be passed in and interpreted at runtime.

**Question 4.1d**
OCaml restricts scope with the use of modules. The scope of a variable depends on the context it is declared, but can, for the most part, be described as being limited to within the module, post-declaration.

**Question 4.1e**
Compiling OCaml in an OO context and running OCaml at the top-level use different scope for a variable. In compiling code, the scope of a variable is dependent on the “in” statement following the let declaration. Variables let’ed at the top-level are considered global and accessible until the top-level’s session termination.

**Question 4.2a**
Global variables do not really exist in OCaml. Rather, what appears to be a global variable is merely a shorthand name for something else. Unlike C and other imperative languages, a “variable” pointer is not stored in memory. Because compilation units imply the use of modules, any variable declared in the root of the compilation unit is not global, per say.

**Question 4.2b**
As said in 4.2a, OCaml does not support global variables in the traditional sense, so this question makes no sense.
**Question 4.2c**

One would want to hide global variables to potentially reuse the name of the variable or to limit the scope of the variable.

**Question 4.3**

If a statement sets aside storage, it is considered a definition. Otherwise, it is considered a declaration. The exception to this is a struct definition (a type definition), which does not allot storage. If no storage is specified, the statement is simply a declaration.

Declaration Example:
int foo;

Definition Example:
int foo = 4;

**Question 4.4**

Header files serve as a place for to store declarations to be used by potentially multiple files. Doing this removes code duplication among other things. Java uses imports to mimic the behavior of header files.