O código para calcular a frequência em Amulet segue.

open import "lua/conversion.ml" (\* 1 \*)  
open import "lua/io.ml" (\* 2 \*)  
open import "prelude.ml"  
  
(\* 3 \*)  
let read\_float () =  
 (\* 4 \*)  
 let str = read\_line (file\_of standard\_in)  
 str >>= parse\_float  
 (\* 5 \*)  
  
let () =  
 let aulas\_assistidas =  
 put\_bytes "Entre o número de aulas assistidas: "  
 (\* 6 \*)  
 match read\_float () with  
 | Some v -> v  
 | None -> error "você não entrou um float válido"  
 let aulas\_totais =  
 put\_bytes "Entre o número de aulas totais: "  
 match read\_float () with  
 | Some v -> v  
 | None -> error "você não entrou um float válido"  
 let freq = aulas\_assistidas /. aulas\_totais (\* 7 \*)  
 print freq  
 if 0.75 <=. freq then (\* 8 \*)  
 put\_line "Aprovado"  
 else  
 put\_line "Reprovado"

1. Comentários são escritos entre parênteses seguidos de asteriscos: (\* isso é um comentário \*). Não temos síntaxe para um comentário de linha.
2. A expressão import "caminho/para/arquivo.ml" tem como efeito ler o arquivo de código que se encontra no caminho especificado e retorna um termo no estrato dos módulos, isso é, uma coleção de definições. Tendo um módulo em mãos, podemos “explodí-lo” no escopo atual usando open ou let open.
3. Definições de função tem como formato

* let nome argumento\_1 ... argumento\_n =  
   ... corpo ...
* Aqui, os argumentos não são limitados a nomes, e podem ser padrões. Por exemplo, o padrão () reconhece o valor que também é escrito (), que corresponde de certa forma ao tipo void em C.

1. Também podemos usar a forma let para definir nomes para variáveis locais. De fato, como Amulet é uma linguagem funcional, a forma descrita no item 3 é uma abreviação para definir uma função literal com um nome.
2. O operador >>=, pronunciado “bind”, sequência computações que podem falhar. A função read\_line tem tipo unit -> option string, o que indica que, dado o valor (), ela retorna uma string “opcional”, ou seja, que pode não estar presente. O tipo option é uma disjunção entre os casos Some e None, que indicam presença e falta respectivamente. Do mesmo modo, a função parse\_float tem tipo string -> option float. Podemos “juntar” a option string com a função string -> option float usando o operador >>=, para criar um único valor de tipo option float.

* Em Amulet, nós usamos um sistema de *type classes* para resolver “overloading” (sobrecarga) estáticamente. Portanto, o operador (>>=) funciona para qualquer tipo que se encontra na classe monad. *Type classes* são, de certo modo, correspondente às interfaces da programação orientada à objetos.

1. A estrutura match serve para reconhecimento de padrões com várias alternativas. Isso é, o valor inspecionado é avaliado, e é comparado, caso-a-caso, com os padrões (escritos à esquerda da flecha ->). A alternativa selecionada é a primeira cujo padrão que reconhece o valor inspecionado.
2. Em Amulet, os operadores numéricos não são sobrecarregados: / funciona para inteiros e /. para floats. De modo geral, o operador X tem uma versão equivalente para floats que é escrita X.: +., -., <=., etc.
3. As estruturas de controle em Amulet são todas expressões, ou seja, tem um valor. Portanto, não é possível escrever um if sem uma cláusula else.

* Internamente, if é convertido em um match sobre o tipo bool:
* if x then  
   ... (\* código then \*)  
  else  
   ... (\* código else \*)
* se torna
* match x with  
  | true -> (\* código then \*)  
  | false -> (\* código else \*)