CSE 341 - Section 2 Exercises (from website code)

1. Type synonyms

Below is an example use of type synonyms, with three example date representations:

```
let fst3 (x,_,_) = x (* gets the first element of a triple *)
let snd3 (_,x,_) = x (* gets the second element of a triple *)
let thd3 (_,_,x) = x (* gets the third element of a triple *)

type date = int * int * int
let date1 : int * int * int = (22, 2, 1900)
let date2 : date = (11, 1, 1900)
let date3 = (17, 1, 2019)
```

A. Without using this `date` type, write a function `dmy_to_mdy` to change the date format from (day, month, year) to (month, day, year). Functions should return `int * int * int`.

B. Now write `dmy_to_mdy2` using the `date` type synonym instead.

C. For each of your two functions, write test calls using the example dates which are valid arguments.

2. Type Generality

A. Review: Write a function `append` that takes two string list arguments xs and ys and returns the result of appending them together.

B. Now change your function to accept any 'a list and return an appended result 'a list. This function should return the same list for the example calls in 2.A.

C. Now consider the following lists:

Consider the following lists:

```
let list1 = ["hi"; "bye"]
let list2 = ["programming"; "languages"]
let list3 = [1; 2]
let list4 = [3; 4; 1]
```

Write three test calls to your function in B - two should be correct (and provide the result) and the third should result in a type error.

3. More variant types and Pattern-matching examples (unlikely to get to this)

Consider the following type and variant type:

```
type cart = float * float
type shape = Circle of cart * float (* coordinates and radius *)
           | Square of cart * float (* coordinates and side length *)
           | Rectangle of cart * float * float (* coordinates and side lengths *)
```

A. Write a function `area` which takes a shape as an argument and returns its area (as a float value).

Now consider this variant type to represent expression trees, which lecture uses too:

```
type exp = Constant of int
         | Negate of exp
         | Add of exp * exp
         | Multiply of exp * exp
```

B. Write an ML function `const_not_under_add` of type `exp -> bool` that returns true if and only if there exists a Constant in the expression that is not a child of an Add expression. For example, `const_not_under_add(Constant 341)` should return true, as should `const_not_under_add(Multiply(Constant 341, Add(Constant 0, Constant 1)))`.

C. On your own: What is another function you can think of using the shape or exp variant type?