Exercise #8:

Will this function work?
Will `drop_value 2 [1;2;3]` evaluate to `[1; 3]`?

```
let rec drop_value to_drop l =
  match l with
  | [] -> []
  | to_drop ::  tl -> drop_value to_drop tl
  | hd ::  tl -> hd ::  drop_value to_drop tl
```

Exercise #9:

Fix this function so that `drop_value [1; 2; 3] 2` evaluates to `[1; 3]`.

```
let rec drop_value to_drop l =
  match l with
  | [] -> []
  | to_drop ::  tl -> drop_value to_drop tl
  | hd ::  tl -> hd ::  drop_value to_drop tl
```

Exercise #10:

Write a function to return the first element of a triple.

What is its type?

Exercise #11:

What type would you use to represent fractions?

Exercise #12:

Consider a function to add two fractions.

What is its type?

What is its value?

Exercise #13:

- A *partial mapping* from, say, strings to integers could be represented by a value of the type `(string * int) list`.
- ```
  let m = [ ("dog", 1); ("chicken", 2);
            ("dog", 3); ("cat", 5 ]
  ```
- `lookup_all "cat" m` evaluates to `[5]`
- `lookup_all "moose" m` evaluates to `[]`
- `lookup_all "dog" m` evaluates to `[1; 3]`
- Write `lookup_all`. What is its type?
  ```

Exercise #14:

Rewrite `fib` to use pattern matching.

`fib` was:

```
let rec fib x =
  if x = 0 then 0 else
    if x = 1 then 1 else fib (x-1) + fib (x-2)
```

Exercise #15:

What other types of errors are can you name?

List as many as can.

Also note when they can be detected? By a compiler, or only at run-time.

```
int
     3                    -10            45
bool
     true                 false
string
     "Hello"              "World"
int -> int
     fun x -> x + 1       fun x -> x * x
int -> string
     fun n -> int_of_string n
int list
     [ 1; 2; 3 ]          [ 4; 5; 6 ]
bool list
     [ true; false ]      [false; false; false ]
int * char
     (1, 'c')             (4, 'z')
(int -> int) list
     [ fun x -> x + 1 ; fun x -> x * x ]
```

Strong static type systems

- OCaml has a *strong*, *static* type system
- It is a *safe* language.
- What does "safe" mean?

Expressiveness of types

- OCaml doesn't detect division by 0, but the hardware will.
- Since the hardware doesn't detect type-incorrect operations or invalid memory accesses (within the users allotted memory space), OCaml must prevent these.
- So OCaml can detect all invalid operations
  - some through the static type system
  - some through dynamic (run-time) checks

Static vs Dynamic Typing

- A *static* type system works at compile time, before the program runs, to detect type errors.
  - Java, C, OCaml, Haskell have static type systems.
- A *dynamic* type system works at program run time, as the program executes.
  - Python, Scheme, Ruby, Clojure have dynamic type systems.

Type systems

The challenge - design strong static type systems that are

1. expressive, and

Type systems

The challenge - design strong static type systems that are

1. expressive, and
   - It is difficult to have a static type for non-zero integers.
   - So the question becomes

     "What properties can types express?"

Type systems

The challenge - design strong static type systems that are

1. expressive, and
   - It is difficult to have a static type for non-zero integers.
   - So the question becomes

     "What properties can types express?"

2. easy to use.
   - Type inference can help with this as we don't need to write down all the types. But it is recommended to write types for some parts to provide machine-checked documentation.

Exercise #16:Operator precedence and associativity

- What is the precedence of `*` compared to `->` ?
  How could we find out?

- What is the associativity of `*` ?
  Does it matter?