CSE216
Foundations of Computer Science

State University of New York, Korea

Exercise 1. (points = 25)

Write a function drop : int -> 'a list -> 'a list such that drop n lst returns all but the first n elements of lst.
If lst has fewer than n elements, return the empty list. Here, n can be any integer including negative number.

Problem 2. Ocaml Types (25)

Give the type of the following OCaml expressions:

(1) `["Annyeonghaseyo"]`

(2) `[["Annyeong"];["haseyo"]]`

(3) `("bar", [1;2])`

(4) `[(1,"foo");(3,"bar")]`

(5) `let f x y z = x+y+z in f 1`

Give the type of the function defined below:

(6) `let f x y = x *. 2.7`

(7) `let f (x,y) = (y,x)`

(8)** `let f x y = x :: y`

(9) `let f (x, y) = [x; y]`

(10) `let f x y z = if z then x else y`

Problem 3 (30)

1. Write an Ocaml function to calculate the nth Fibonacci number. Note the Fibonacci numbers look like this starting from the first one: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987.

2. Write an Ocaml function that takes a list of integers and returns a list of the squares of those integers.

3. Write an Ocaml function `compress` that eliminates consecutive duplicates of list elements.

```
# compress ["a"; "a"; "a"; "a"; "b"; "c"; "c"; "a"; "a"; "d"; "e"; "e";
"e"; "e"];;
- : string list = ["a"; "b"; "c"; "a"; "d"; "e"]
```

Problem 4 (20)

Natural numbers are inductively defined as the following grammar

```
n -> Z | S n
```

where Z can be understood as zero, S can be understood the operator that increments by one.

1. Create a type `nat` following this definition of natural number.

2. Define a function

```
add : nat-> nat->nat
```

that adds two natural numbers.

3. Define a function

```
mul : nat-> nat->nat
```

that multiplies two natural numbers.

Bonus

Point = 10

Some programming languages (like Python) allow us to quickly *slice* a list based on two integers `i` and `j`, to return the sublist from index `i` (inclusive) and `j` (not inclusive). We want such a slicing function in OCaml as well.

Write a function `slice` as follows: given a list and two indices, `i` and `j`, extract the slice of the list containing the elements from the $\texttt{i}^{\text{th}}$ (inclusive) to the $\texttt{j}^{\text{th}}$ (not inclusive) positions in the original list.

```
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 2 6;;
- : string list = ["c"; "d"; "e"; "f"]
```

Invalid index arguments should be handled *gracefully*. For example,

```
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 3 2;;
- : string list = []
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 3 20;
- : string list = ["d";"e";"f";"g";"h"];
```

You do *not*, however, need to worry about handling negative indices.