Lab 9

Problem 1

- Evaluate expressions:

```
2 + 3;;
(* int = 5 *)

4.0 +. 5.0;;
(* float = 9. *)

4.0 +.  5;;
(* Error *)
```

Problem 2

- Evaluate expressions:

```
'a';;
(* char = 'a' *)

"the result" ^ " is " ^ "a string";;
(* string = "the result is a string" *)

true && false;;
(* bool = false *)

false || not (2 = 3);;
(* bool = true *)
```

Problem 3
- Evaluate expressions:

```
5 < 6;;
(* bool = true *)

"beta" > "alpha";;
(* bool = true *)

[];;
(* 'a list = [] *)

[1; 2; 3];;
(* int list = [1; 2; 3] *)

1::2::3::[];;
(* int list = [1; 2; 3] *)

['a'; 'b'] @ ['c'; 'd'; 'e'];;
(* char list = ['a'; 'b'; 'c'; 'd'; 'e'] *)
```

Problem 4
- Fibonacci

```
let rec fib = fun n ->
    if n <= 1 then
        n
    else
        fib (n - 1) + fib (n - 2)
;;

let rec fib_seq = fun n ->
    if n <= 1 then
        [n]
    else
        fib_seq (n - 1) @ [fib n]
;;
```

- Test Cases:

```
fib 13;;
(* int = 10946 *)
fib 5;;
(* int = 5 *)

fib_seq 13;;
(* int list = [1; 1; 2; 3; 5; 8; 13; 21; 34; 55; 89; 144; 233]
*)
fib_seq 5;;
(* int list = [1; 1; 2; 3; 5] *)
```

Problem 5

- Ackermann function

```
let rec ack = fun x y ->
    if x = 0 then
        (y + 1)
    else if y > 0 then
        ack (x - 1) (ack x (y - 1))
    else
        ack (x - 1) 1
;;
```

- Test Cases:

```
ack 2 1;;
(* int = 5 *)

ack 1 2;;
(* int = 4 *)

ack 3 2;;
(* int = 29 *)

ack 3 3;;
(* int = 61 *)

ack 3 6;;
(* int = 509 *)
```

Problem 6

- Generate Interval

```
let rec geninterval = fun a b ->
    if a = b then
        [a]
    else
        a :: geninterval (a + 1) b
;;
```

- Test Cases:

```
geninterval 1 10;;
(* int list = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10] *)

geninterval (-2) 2;;
(* int list = [-2; -1; 0; 1; 2] *)
```

Problem 7

- Not

```
let my_not = fun cond ->
    if cond then
        false
    else
        true
;;
```

- And

```
let my_and = fun cond1 cond2 ->
    if cond1 then
        cond2
    else
        false
;;
```

- Or

```
let my_or = fun cond1 cond2 ->
    if cond1 then
        true
    else
        cond2
;;
```

- Test Cases:

```
Printf.printf "my_not(T) = %b\n" (my_not true);;
Printf.printf "my_not(F) = %b\n" (my_not false);;
(* my_not(T) = false *)
(* my_not(F) = true *)

Printf.printf "my_and(T, T) = %b\n" (my_and true true);;
Printf.printf "my_and(T, F) = %b\n" (my_and true false);;
Printf.printf "my_and(F, T) = %b\n" (my_and false true);;
Printf.printf "my_and(F, F) = %b\n" (my_and false false);;
(* my_and(T, T) = true  *)
(* my_and(T, F) = false *)
(* my_and(F, T) = false *)
(* my_and(F, F) = false *)

Printf.printf "my_or(T, T) = %b\n" (my_or true true);;
Printf.printf "my_or(T, F) = %b\n" (my_or true false);;
Printf.printf "my_or(F, T) = %b\n" (my_or false true);;
Printf.printf "my_or(F, F) = %b\n" (my_or false false);;
(* my_or(T, T) = true  *)
(* my_or(T, F) = true  *)
(* my_or(F, T) = true  *)
(* my_or(F, F) = false *)
```

Problem 8
- Digits

```
let rec digits = fun num ->
    if num = 0 then
        []
    else
        digits (num / 10) @ [(num mod 10)]
;;
```

- Test Cases:

```
digits 54281;;
(* int list = [5; 4; 2; 8; 1] *)

digits 1001;;
(* int list = [1; 0; 0; 1] *)

digits 12345678901234567 89;; (* 19 digits *)
(* int list = [1; 2; 3; 4; 5; 6; 7; 8; 9; 0; 1; 2; 3; 4; 5; 6;
7; 8; 9] *)

digits 123456789012345678 90;; (* 20 digits *)
(* Error - out of range of type int *)
```

Problem 9

- Maximum Element 1

```
let rec maximum = fun list ->
    if list = [] then
        Int.min_int
    else
        (max (List.hd list) (maximum (List.tl list)))
;;
```

- Test Cases:

```
maximum [1; 2; 3; 4; 5; 2; 3; 1];;
(* int = 5 *)

maximum [3];;
(* int = 3 *)

maximum [];;
(* int = Int.min_int (-4611686018427387904) *)
```

- Maximum Element 2

```
let rec maximum = fun list ->
    if List.length list = 1 then
        List.hd list
    else
        (max (List.hd list) (maximum (List.tl list)))
;;
```

- Test Case:

```
maximum [1; 2; 3; 4; 5; 2; 3; 1];;
(* int = 5 *)

maximum [3];;
(* int = 3 *)

maximum [];;
(* Exception: (Failure hd) *)
```

Problem 10
- Greatest Common Divisor

```
let rec gcd = fun a b ->
    if a = b then
        a
    else if a < b then
        gcd a (b - a)
    else
        gcd (a - b) b
;;
```

- Test Cases:

```
gcd 36 48;;
(* int = 12 *)

gcd 103 4501;;
(* int = 1 *)

gcd 234 24;;
(* int = 6 *)
```