COL100: Lab 3 Solutions

In case of any error please contact Praveen Kulkarni at `cs5140599@cse.iitd.ac.in`.

fizzbuzz.ml

```
(* Author: Praveen Kulkarni
 * Date: 13 March 2018
 * File: fizzbuzz.ml
 * All rights reserved. Copyright (c) 2018
 *)

(* fizzbuzz : int -> string *)
let fizzbuzz index =
    if index mod 15 = 0 then "Fizzbuzz"
    else if index mod 3 = 0 then "Fizz"
    else if index mod 5 = 0 then "Buzz"
    else (string_of_int index);;

(* tests *)
let _ = fizzbuzz 17;;
let _ = fizzbuzz 18;;
let _ = fizzbuzz 20;;
let _ = fizzbuzz 30;;

(* fizzbuzz_string : int -> string *)
let rec fizzbuzz_string index =
    if index <= 0 then ""
    else if index = 1 then "1"
    else (fizzbuzz_string (index-1)) ^ " " ^ (fizzbuzz index);;

(* tests *)
let _ = fizzbuzz_string 4;;
let _ = fizzbuzz_string 10;;
let _ = fizzbuzz_string 15;;
```

leap_year.ml

```ocaml
(* Author: Praveen Kulkarni
 * Date: 13 March 2018
 * File: leap_year.ml
 * All rights reserved. Copyright (c) 2018
 *)

(* isLeapYear: int -> bool *)
let isLeapYear year =
    if (year mod 4 != 0) then false
    else if (year mod 100 != 0) then true
    else (year mod 400 = 0);;

(* testcases *)
let _ = isLeapYear 2004;;
let _ = isLeapYear 2016;;
let _ = isLeapYear 2000;;
let _ = isLeapYear 2017;;
let _ = isLeapYear 2018;;
let _ = isLeapYear 1900;;
```

middle_child.ml

```
1  (* Author: Praveen Kulkarni
2   * Date: 13 March 2018
3   * File: middle_child.ml
4   * All rights reserved. Copyright (c) 2018
5   *)
6
7  (* middleChild : int -> int -> int -> bool *)
8  let middleChild x y z =
9      if (x < 0 || y < 0 || z < 0) then -3
10     else if (x = y && y = z) then -2
11     else if (x = y || y = z || x = z) then -1
12     else (x + y + z) - (max (max x y) z) - (min (min x y) z);;
13
14 (* test cases *)
15 let _ = middleChild 17 12 15;;
16 let _ = middleChild 3 3 5;;
17 let _ = middleChild 12 12 12;;
18
19 (* print_middle_child : int -> int -> int -> string *)
20 let print_middle_child x y z =
21     let middle_child_value = middleChild x y z
22     in
23         if middle_child_value = -3 then "Invalid inputs!"
24         else if middle_child_value = -2 then "There are triplets"
25         else if middle_child_value = -1 then "There are twins!"
26         else "The age of the middle child is:" ^ (string_of_int middle_child_value);;
27
28 (* test cases *)
29 let _ = print_middle_child 17 12 15;;
30 let _ = print_middle_child 3 3 5;;
31 let _ = print_middle_child 12 12 12;;
32 let _ = print_middle_child (-1) 12 12;;
33
34 (* Notice that when negative numbers are passed as arguments you
35  * should put paranthesis around them to avoid ambiguity (line 32)
36  *)
```

p_checker.ml

```
(* Author: Praveen Kulkarni
 * Date: 13 March 2018
 * File: p_checker.ml
 * All rights reserved. Copyright (c) 2018
 *)

(* EDITORIAL
```

```
 * ==========
 * There are two functions in this file - check_prime and isPrime.
 *
 * isPrime: int -> bool
 * A number `num` is NOT prime if and only if there is a number x, such that
 * 2 <= x <= sqrt(num).
 * If num is less than 2 then it is not prime, so isPrime returns false.
 * If num is equal to 2, then it is prime, so isPrime returns true.
 * If num is greater than 2, then we have to search for an x from 2 to sqrt(x).
 *
 * For that we have built a recursive function `check_prime`.
 *
 * To understand recursion, you must use inductive proofs.
 * =========================================================
 * Intuition:
 * Assume that a number `num` is not divisible by 2, 3, ...., x-1.
 * Then three cases can happen: -
 * 1.   If x * x > num, then num is PRIME. Because no number from 2 ...x
 *      divided x, then no number in the range x+1 ... num-1 can possibly
 *      divide num.
 * 2.   If x divides num, then num is NOT PRIME, by definition. Again we get a
 *      result, so we can stop.
 * 3.   If x doesn't divide num, then we can now strengthen our assumption that
 *      `num` is not divisible by 2, 3, .... x-1, x.
 *
 * We can start with the assumption that the num > 2. We should start with x=2,
 * and apply the procedure above. If we repeat it enough number of times, then
 * we will come to an answer whether num is prime or not.
 *
 * This is exactly what the `check_prime` function does.
 * When `check_prime num x` is called, then we have some guarantees,
 * 1.   num is an integer > 2.
 * 2.   num is not divisible by any integer y such that 1 < y < x.
 *
 * From the intuition above, you should be able to write an inductive proof, why
 * `check_prime num 2` works.
 *)

(*  check_prime : int -> int -> bool *)
let rec check_prime num x =
    if (x * x > num) then true
    else if num mod x = 0 then false
    else (check_prime num (x+1));;

(* isPrime: int -> bool *)
let isPrime num =
    if num <= 1 then false
    else if num = 2 then true
```

```
    else (check_prime num 2);;

let _ = isPrime 0;;
let _ = isPrime 1;;
let _ = isPrime 2;;
let _ = isPrime 3;;
let _ = isPrime 4;;
let _ = isPrime 25;;
let _ = isPrime 97;;
```

square_root.ml

```
(* Author: Praveen Kulkarni
 * Date: 13 March 2018
 * File: square_root.ml
 * All rights reserved. Copyright (c) 2018
 *)

(* newton: float -> float -> float *)
let rec newton a x1 times =
    if times = 0 then x1
    else
        let x2 = (x1 +. (a /. x1)) /. 2.0
        in (newton a x2 (times-1));;

(* square_root : float -> int -> float *)
let square_root num steps =
    if steps <= 0 then (newton num 1.0 20)
    else (newton num 1.0 steps);;

(* tests *)
let _ = square_root 4.0 2;;
let _ = square_root 4.0 0;;
```