CISS445: Programming Languages
Assignment 4

OBJECTIVES

This is the first of several OCAML assignments where the main object is to help you learn the basic OCAML language, including basic types and operators, tuples and lists with their operations, declarations, recursion, and matchings.

This is an easy assignment. Therefore no discussion is allowed.

You MUST refer to a02 for general assignment instructions.

Q1. Write a function `duplicate` so that `(duplicate x n)` will return a list with `n` copies of `x`. Ignore the case where `n` is negative.

Tests	Expected value
`duplicate 1 3`	`[1; 1; 1]`
`duplicate 1.23 0`	`[]`
`duplicate 3.1 4`	`[3.1; 3.1; 3.1; 3.1]`
`duplicate [1;2;3] 2`	`[[1;2;3]; [1;2;3]]`

Q2. Write a function `range` so that `(range a b)` will return a list of integers from `a` to `b - 1`.

Tests	Correct values
`range 1 3`	`[1; 2]`
`range 1 1`	`[]`
`range (-3) 2`	`[-3; -2; -1; 0; 1]`
`range 6 3`	`[]`

Q3. Write a function `range2` so that `(range2 a b c)` returns a list of integers from `a` to `< b` in steps of `c` if `c` is positive and from `a` to `> b` if `c` is negative.

Tests	Expected Value
`range2 1 3 1`	`[1; 2]`
`range2 1 7 2`	`[1; 3; 5]`
`range2 1 6 2`	`[1; 3; 5]`
`range2 1 1 2`	`[]`
`range2 3 (-2) (-1)`	`[3; 2; 1; 0; -1]`
`range2 3 (-2) (-2)`	`[3; 1; -1]`

Q4. Write a function `slice` so that `(slice list a b c)` returns a list consisting of the elements of list from index position `a` to `b - 1` in steps of `c`. (Ignore the case where `c` is negative).

Tests	Expected value
`slice [6;7;8;9] 0 2 1`	`[6; 7]`
`slice [6;7;8;9] 0 3 2`	`[6; 8]`
`slice [6;7;8;9] 1 3 2`	`[7]`
`slice [6;7;8;9] 1 100 100`	`[7]`
`slice [6;7;8;9] 3 1 1`	`[]`

Q5. Write a function index so that (index list x) returns the index position of the first occurrence of x in the list. (-1) is returned if x is not found.

Tests	Expected value
`index [6; 7; 8; 9] 3`	-1
`index [6; 7; 8; 9] 10`	-1
`index [9; 8; 7; 6] 9`	0
`index [9; 7; 6; 8] 7`	1
`index [9; 8; 6; 7] 7`	3
`index [] 1`	-1
`index [1; 2; 3; 3; 3] 3`	2
`index [1.1; 2.2; 3.3; 3.3; 3.3] 3.3`	2

Q6. Write a function `at` so that `(at list n)` returns the element of the list at index `n`. Ignore the case where `n < 0` or `n >=` the size of the list.

Tests	Expected values
`at [5;3;1] 0`	5
`at [2;4;6] 1`	4
`at [1;3;5;7] 3`	7

Q7. Write a function `elementof` such that `(elementof x list)` returns true if `x` is a value in `list`.

Tests	Expected value
`elementof 1 [1; 3; 1; 5; 3; 5]`	`true`
`elementof 3 [1; 3; 1; 5; 3; 5]`	`true`
`elementof 5 [1; 3; 1; 5; 3; 5]`	`true`
`elementof 0 [1; 3; 1; 5; 3; 5]`	`false`
`elementof 2 [1; 3; 1; 5; 3; 5]`	`false`
`elementof 4 [1; 3; 1; 5; 3; 5]`	`false`
`elementof 6 [1; 3; 1; 5; 3; 5]`	`false`
`elementof 3.4 [1.2; 3.4; 5.6]`	`true`
`elementof 7.8 [1.2; 3.4; 5.6]`	`false`

Q8. Write a function `subseteq` such that `subseteq list1 list2` is true iff every value in `list` is in `list2`.

Tests	Expected value
`subseteq [] [1; 3; 5]`	`true`
`subseteq [] []`	`true`
`subseteq [2; 2; 2] [1; 2; 3]`	`true`
`subseteq [3; 2; 1] [1; 2; 3]`	`true`
`subseteq [1; 3; 5] [3; 1; 5; 7]`	`true`
`subseteq [1; 3; 5] [1; 3]`	`false`