Large Assignment #1: SML
Due: Friday, 11 October 2024 by 11:59 PM

Instructions.
Implement the functions below, following all the guidelines in a file called `la1.sml` and submit it on lectura using the following command:

```
turnin csc372la1 la1.sml
```

General Guidelines.

- You are only allowed to use the SML features discussed in class or provided in the notes on D2L.
- You should use pattern matching and minimize the use of type annotations and conditionals.
- Any helper functions should be in let blocks (unless it is the answer to a previous question).
- Put multiple function definitions on separate lines.
- Some of the functions have individual requirements. Make sure you follow those.
- If any functions are incomplete and/or do not compile, comment them out.
- Double-check the spelling of the function names and the types.
- Put the number of the question in a comment above the function implementation.

Academic Integrity.
If you are found to have violated the academic integrity policy, you will be reported to the Dean of Students office and the recommended sanction will be a failing grade in the course.

Grading.
These are graded for accuracy and style. The points for each function are listed in the table below. These will be graded on lectura, so make sure you test it there before submission.

Required Functions

#	Function Name	Type	Description	Points
1	`triangle`	`int*int*int → bool`	`The triangle inequality theorem states that the sum of any two sides of a triangle must be greater than`	1

			or equal to the third side. This function should return true if the three integers can make a triangle and false otherwise.	
2	triangleR	real*real*real → bool	The triangle inequality theorem states that the sum of any two sides of a triangle must be greater than or equal to the third side. This function should return true if the three real numbers can make a triangle and false otherwise.	1
3	cycle	int*\`a list → \`a list	Given an integer n and a list l, cycle n elements from the list to the end of the list. You can assume the input for n will be non-negative. Example: cycle(4,[1,2,3,4,5,6,7]) → [5,6,7,1,2,3,4]	2
4	mirror	\`a list → \`a list	Mirror the list. You may not use any reverse function (even as a helper function). Example: mirror [1,2,3,4] → [1,2,3,4,4,3,2,1]	2
5	gtList	int list * int → int list	Take a list l and an integer n and return a list that contains all the elements in l that are greater than n. Keep the same relative order of items.	2
6	suffix	\`\`a list * \`\`a list → bool	Return true if the first list is a suffix of the second list and false otherwise. Do not reverse either of the lists.	2
7	get	\`a list * int → \`a	This function takes a list and an integer i and returns the i^{th} element in the list. Start the indexing at 0.	2
8	subList	\`a list * int * int → \`a list	This function takes a list and two integers i and j and returns a subList that includes everything from the i^{th} to the j^{th} element (inclusive). (Hint: you may want to use the get function from #7 and since it is required for #7, it does not need to be included in a	4

			let block here.)	
9	reverse	\`a list → \`a list	Reverse the list *without using append.*	3
10	apply	\`a list * (\`a → \`b) → \`b list	Apply a function to a list of elements. You may not use map, foldr, or foldl. Examples: apply([(1,2),(3,4),(5,6)],(op +)) → [3,7,11] apply((explode “hello”),ord) → [104,101,108,108,111]	3
11	collapse	\`a list * \`b * (\`a * \`b → \`b) → \`b	Take a list, a starting value, and a function and collapse the list down using the function. You may not use map, foldr, or foldl. Examples: collapse([1,2,3,4,5],0,(op +)) → 15 collapse([1.1,2.2,3.3],10.0,(op -)) → ~7.8	3
12	quicksort	(\`a * \`a → bool) → \`a list → \`a list	Sort a list using the quicksort method, but make it general enough so that you can pass in a function and sort in either direction and/or sort various types of data.	4
13	bubbleSort	(\`a * \`a → bool) → \`a list → \`a list	Sort a list using the bubble sort method, but make it general enough so that you can pass in a function and sort in either direction and/or sort various types of data.	4
14	insertionSort	(\`a * \`a → bool) → \`a list → \`a list	Sort a list using the insertion sort method, but make it general enough so that you can pass in a function and sort in either direction and/or sort various types of data.	4
15	substring	string → string → bool	Determine if a string is a substring of another string.	3

16	indexOf	``a → ``a list → int	Determine the index of the first occurrence of a value in a list. Indexing should start at 0.	2
17	dec2BaseN	int → int → string	Turn a decimal number into a string representation of base N. The input for N will be an integer between 2 and 10 (inclusive).	3
18	dropNth	int → `a list → `a list	Drop every nth element in a list.	3
19	flatten	`a list list → `a list	Takes a list of lists and flattens it so that it is now just a single list of elements. It should contain all the original elements in the same order, but they should no longer be separated into lists. This should be a one-liner using map, foldr, and/or foldl.	2
20	condenseLists	(`a * `b → `b) → `b → `a list list → `b list	Take a list of lists, a function, and a starting value. Apply the function recursively to each list, condensing it to a single value. The result is the list of the results for each inner list. This should be a one-liner using map, foldr, and/or foldl.	2
21	remove	(`a → bool) → `a list → `a list	Remove all the elements from a list for which the given function returns true. This should be a one-liner using map, foldr, and/or foldl.	2
22	triplist	'a list → 'a list	Take a list and create a new list where every element is repeated three times. Do not use append.	2
23	repeat	'a list → int → 'a list	Take a list l and an integer n and create a list that repeats l n times.	3
24	filterApply	'a list → ('a → bool) → ('a → 'a)	Takes a list l, a function f, and a function g. It then returns a list where all the elements in l for which f is true have g applied to them. You should do	2

		→ 'a list	this in one line with map, foldr, and/or foldl.	
25	arithSeq	int → int → int → int list	Given three integers (a, d, and l), create a list of length l that contains an arithmetic sequence starting at a with a common difference of d. Do not use append or a reverse list function.	3
26	element	``a → ``a list → bool	Return true if the element is in the list and false otherwise.	2
27	isSet	``a list → bool	Return true if the list is a set, meaning that it has no duplicate values. Return false otherwise.	2
28	union	``a list * ``a list → ``a list	Combine two lists, which you can assume are sets, to produce the union of the two sets. The resulting order of the elements does not matter.	2
29	intersection	``a list * ``a list → ``a list	Combine two lists, which you can assume are sets, to produce the intersection of the two sets. The resulting order of the elements does not matter.	2
30	difference	``a list * ``a list → ``a list	Combine two lists, which you can assume are sets, to produce the difference of the two sets (i.e. the items that are in the first set but not in the second set.) The resulting order of the elements does not matter.	2
31	xor	``a list * ``a list → ``a list	Combine two lists, which you can assume are sets, to produce the xor of the two sets (i.e. any items that are in one set but not the	2

			other.) The resulting order of the elements does not matter.	
32	powerset	`a list → `a list list	Given a list that you can assume is a set, return the powerset of the set (i.e. the set of all possible subsets.)	4