This functional programming assignment requires you to represent finite sets in two different ways, using *two different*data types:  (a) OCaml lists, and (b) OCaml functions:

(a) a set is represented as a ***list without duplicates***.

You need to implement the following operations, and you should use list functions such as map, fold, filter etc.

1. **emptyset**, which represents the empty set.
2. **member** *x s*, which returns true if and only if *x* is in *s*.
3. **subset** *s1 s2*, which returns true if and only if *s1* is a subset of *s2*
4. **equal** *s1 s2*, which returns true if and only if  set *s1* is equal to set *s2*.
5. **union** *s1 s2*, which returns the union of sets *s1* and *s2*
6. **intersection** *s1 s2*, which returns the intersection of*s1* and *s2*
7. **difference** *s1 s2*, which returns the set consisting of elements of *s1* which are not in *s2*
8. **power** *s*, which returns the set of subsets of *s*
9. **product** *s1 s2*, which returns the cartesian product of *s1* and *s2*.

(b) a set is represented by its **characteristic function** -- *x*\in *s* iff *f\_s (x)*= true (where*f\_s* is the characteristic function of set *s*)

1. **emptyset**, which represents the empty set.
2. **member** *x s*, which returns true if and only if *x* is in *s*.
3. **union** *s1 s2*, which returns the union of sets *s1* and *s2*
4. **intersection** *s1 s2*, which returns the intersection of*s1* and *s2*
5. **difference** *s1 s2*, which returns the set consisting of elements of *s1* which are not in *s2*
6. **product** *s1 s2*, which returns the cartesian product of *s1* and *s2*.

Think how you might be able to implement

* **subset** *s1 s2*, which returns true if and only if *s1* is a subset of *s2*
* **equal** *s1 s2*, which returns true if and only if  set *s1* is equal to set *s2*.
* **power** *s*, which returns the set of subsets of