COL100 Assignment 4

Due date: 5 February 2022, 23:59

Instructions: All programs are to be written in Python. Functions must be documented with proper comments stating

- the purpose of the function
- the types and meanings of the input arguments
- any conditions on the input arguments (input specifications)
- the type and meaning of the return values
- conditions on the outputs (output specifications)

Within the program you MUST document the following:

- Representation invariant that your function maintains stated (at least informally) as a comment, and a check that this is maintained. For example, if the input arguments are assumed to contain no duplicates, there should be no duplicates in the output.
- Loop invariant for each loop, even if informally stated, which give the intuition about why the program is correct.
- Highlight (via a comment) a decreasing measure in each while loop which guarantees termination of the loop.
- Time complexity of the function, with at least an informal justification.
- Representative test data on which you have run the function.
- Use appropriate notation in your arguments/explanations. You will get 0 for not writing appropriate explanations. You will be asked to explain these during the demo.

Part 1

In the first part of assignment you will develop a package for sets and their manipulation using operations such as union, intersection, Cartesian product etc.

You have to implement a set S over elements of some type **as a list that contains no duplicates**. (Here the type T can variously be integer, floats, or even a list of some type).

Note that
1. Set $\{1, 2, 3\}$ can be represented as [1,2,3] but not [1,2,2,3].
2. Ordering of elements does not matter. [1,2,3] and [2,3,1] are same.
3. There are many possible outputs of Q7 to Q10 in part1 which are different representation of same set, we have wrote here just one of them.

Implement the following functions for your implementation of sets. These functions represent the basic set operations.

1. A function `emptyset()`, which returns the representation of an empty set.

2. A boolean function `isEmpty`(S), returning *True* if S is empty and *False* otherwise.

3. A function `member`(S, e) where s is a set of elements of some type T which represented as a list and e is an element of that type. If e is a member of S, `member` should return *True* and otherwise should return *False*. Example

   ```
   >>> member([4,2,3,1],4)
   True
   >>> member(["ELL100","MTL100","COL100"],"MTL100")
   True
   ```

4. A function `singleton`(x), which returns the representation of a singleton set consisting of only the element x.

5. A function `isSubset`(P, Q), which given representations of two sets (of the same type) P and Q, returns *True* if P is a subset of Q and *False* otherwise. Example

```
>>> isSubset([1,2,3],[3,2,1,4])
True
```

6. A function `setEqual`(P, Q), which given representations of two sets (of the same type) P and Q, returns *True* if P and Q have the same elements, and *False* otherwise. Example

```
>>> setEqual([1,2,3],[3,2,1])
True
```

7. A function `union`(P, Q), where P, Q are (representations of) two sets (of same type T), which returns the representation of the set $P \cup Q$. Example

```
>>> union(["MTL100","COL100"],["NLN100","MTL100"])
["MTL100","COL100","NLN100"]
```

8. A function `intersection`(P, Q) where P, Q are (representations of) two sets (of same type T), which returns the representation of the set $P \cap Q$.

```
>>> intersection(["MTL100","COL100"],["NLN100","MTL100"])
["MTL100"]
```

9. A function `cartesian`(P, Q), where P, Q are (representations of) two sets (not necessarily of same type), which returns the representation of the set $P \times Q$.

```
>>> cartesian([1,3,2],["C","M"])
[(1,"C"),(2,"M"),(3,"C"),(1,"M"),(2,"C"),(3,"M")]
```

10. A function `power`(P), where P is (the representation of) some set, which returns the representation of the set of *all* subsets of P. Example

```
>>> power([1,3,2])
[[], [1], [3], [3, 1], [2], [2, 1], [2, 3], [2, 3, 1]]
```

Part 2

Next assume that the elements are drawn from a *totally ordered* set, *e.g.* integers or strings. Assume we represent sets as *ordered lists* of elements, according to the total order on elements of that type.
Note that
1. Ordering of element matter. [1,2,3] is the only allowed representation of the set {1,2,3}. The list [2,3,1] is not allowed because it is not sorted.
2. You can assume input sets are ordered. You have to ensure the output sets are also ordered.
Implement the following functions to give more efficient implementation (which can still be in the worst case be as inefficient).

1. A function `emptyset_2()`, which returns the representation of an empty set.

2. A boolean function `isEmpty_2`(S), returning *True* if S is empty and *False* otherwise.

3. A function `member_2`(S, e) where s is a set of elements of some type T which represented as a list and e is an element of that type. If e is a member of S, `member` should return *True* and otherwise should return *False*. Example

   ```
   >>> member_2([1,2,3,4],4)
   True
   ```

4. A function `singleton_2`(x), which returns the representation of a singleton set consisting of only the element x.

5. A function `isSubset_2`(P, Q), which given representations of two sets (of the same type) P and Q, returns *True* if P is a subset of Q and *False* otherwise. Example

   ```
   >>> isSubset_2([1,2,9],[2,3,6])
   False
   ```

6. A function `setEqual_2`(P, Q), which given representations of two sets (of

the same type) P and Q, returns $True$ if P and Q have the same elements, and $False$ otherwise. Example

```
>>> setEqual_2([1,2,3],[1,2])
False
>>> setEqual_2([1,2,3],[1,2,3])
True
```

7. A function `union_2`(P, Q), where P, Q are (representations of) two sets (of same type T), which returns the representation of the set $P \cup Q$. Example

```
>>> union_2(["COL","ELL","MTL"],["MTL","PYL"])
["COL","ELL","MTL","PYL"]
```

8. A function `intersection_2`(P, Q) where P, Q are (representations of) two sets (of same type T), which returns the representation of the set $P \cap Q$. Example

```
>>> intersection_2([1,2,3,4],[3,4,5,6,7,8])
[3,4]
```

9. A function `cartesian_2`(P, Q), where P, Q are (representations of) two sets (not necessarily of same type), which returns the representation of the set $P \times Q$. Example

```
>>> cartesian_2([1,2,3],["C","M"])
[(1,"C"),(1,"M"),(2,"C"),(2,"M"),(3,"C"),(3,"M")]
```

10. A function `power_2`(P), where P is (the representation of) some set, which returns the representation of the set of *all* subsets of P.

```
>>> power_2([1,2,3])
[[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]
```

Check that the Cartesian product and power set operations satisfy the representational invariant (i.e., check whether the Cartesian product of two sets can be represented as an ordered list over $A \times B$, and indicate what the ordering relation is. Similarly for the power set of a given set.

Again, all the expected documentation should be given.

[Important Note: In this assignment you have to focus more on clear explanations conveying your understanding than on coding.]

Related Links

Moodle course: Here

Queries / FAQs: Here

Submission Instructions

You should keep in mind the following instructions while making your submission:

1. The name of your file should be in this format: **⟨entry_number⟩_assignment_4.py**
2. Keep in mind, marks would be deducted if your submission is not a python file i.e. without a .py extension or with an inconsistent filename.
3. You don't need to zip your files.
4. Submission is to be made over moodle.
5. Further instructions will be provided here.