TUTORIAL EXERCISES #1

EXPLORING A NEW STATEMENT FORM

A statement is a sentence that is evaluated to either true or false. Consider the statement:

(S1) Only pernicious people are quixotic

Let’s study (S1) in various ways, without worrying about the meaning of pernicious or quixotic. The goal is to practice different ways of thinking about statements, making their meaning more explicit to you. This is especially important when reading or making long sequences of statements: any hesitation and uncertainty about the individual statements starts to really add up and interfere.

For each approach below, try to see how much you can do from statement (S1) as given (as opposed to rewriting it). Afterwards, check that your answers in each section are consistent with each other. Feel free to skip around: there’s no particular order.

DATABASE. What would a relevant database, or table, look like?
Give example databases where (S1) is true and ones where it’s false.
How many significantly different example databases can you make?
To check that a database has the property, do you restrict yourself to certain entries, and check a certain property? Which ones, and which property?

DIAGRAM. Draw an appropriate diagram (Venn diagram) for exploring (S1).
Can you describe (S1) in terms of the diagram? Show where members satisfying (S1) may be located and where members falsifying (S1) may be located.

COUNTEREXAMPLE. Can (S1) be disproven by a counterexample? If so, what property or properties would a counterexample have?

MEANING FOR INDIVIDUALS. Does (S1) make a claim about:

(1) A pernicious person?
(2) A non-pernicious person?
(3) A quixotic person?
(4) A non-quixotic person?

PROGRAMMING. Write a Python function that checks (S1) (you’ll need to use some judgement about modeling various parts of this project). Assume that methods for checking whether a person is pernicious or quixotic have already been written, as well as a list of persons.
As a variation, assume that P is a python list of all pernicious people, and that Q is a python list of all quixotic people.
Write a python function that takes P and Q as arguments, and returns True exactly when S1 is true.

GEOGRAPHY. Which, if any, of the following matches (S1):

(1) The city of pernicious people is in the country of quixotic people.
(2) The city of quixotic people is in the country of pernicious people.
Does this image make any of the earlier questions easier for you, or at least quicker to answer?

FAMILIAR PROPERTIES. You may find it easier to reason about a statement like (S1) when you substitute more familiar properties:

(S2) Only happy students take CSC165

How quickly can you answer all the questions about (S1) for (S2) instead?
More Universal Quantification. Write

(S3) All ribald people are salacious
in the form of (S1).

1. Some CSC108 Logic Review

(1) Is there a Python expression equivalent to
not (a and b)
that doesn’t use and?

(2) Is there a Python expression equivalent to
not (a or b)
that doesn’t use or?
Database. What would a relevant database, or table, look like?

A table of people, with each row containing an identifier (name?) for the person, and columns entries indicating whether or not they are pernicious and whether or not they are quixotic.

Give example databases where (S1) is true and ones where it’s false.

A counter-example to S1 is a quixotic person who is not pernicious. Any database without such a person is one where S1 is true. A database with such a person falsifies S1.

How many significantly different example databases can you make?

With respect to these properties, there are four types of people: those who are both quixotic and pernicious, those who just quixotic, those who just pernicious, and those who are neither. We can classify databases according to whether they have, or don’t have, each sort of person. One terse classification would be a 4-bit binary string, with a 1 in position if there is at least one person who is both quixotic and pernicious, and a zero in position one otherwise. A database that has people who are both quixotic and pernicious, as well as people who are just quixotic, but nobody who is just pernicious or who is neither pernicious nor quixotic would be classified with the string 1100.

Since there are sixteen 4-bit binary strings, there are sixteen significantly different example databases.

To check that a database has the property, do you restrict yourself to certain entries, and check a certain property? Which ones, and which property?

I can restrict my attention to quixotic people, and check that they are also pernicious.

Diagram. Draw an appropriate diagram (Venn diagram) for exploring (S1).

Draw a rectangle representing the universe (people, probably), containing two interlocking circles representing pernicious and quixotic people, respectively.

Can you describe (S1) in terms of the diagram? Show where members satisfying (S1) may be located and where members falsifying (S1) may be located.

The diagram has four distinct regions. The only configuration that falsifies S1 is the one in which the region representing quixotic people who are not pernicious is non-empty. All other configurations are consistent with S1.

Counterexample. Can (S1) be disproven by a counterexample? If so, what property or properties would a counterexample have?

A counterexample to S1 is a quixotic person is who not pernicious. The existence of such a counterexample disproves S1.

Meaning for individuals. Does (S1) make a claim about:

1. A pernicious person? No, they may be quixotic, or non-quixotic.
2. A non-pernicious person? Yes, they must be non-quixotic according to S1.
3. A quixotic person? Yes, they must be pernicious according to the contrapositive of S1.
4. A non-quixotic person? No, they may be pernicious, or non-pernicious.

Programming. Write a Python function that checks (S1) (you’ll need to use some judgement about modeling various parts of this project). Assume that methods for checking whether a person is pernicious or quixotic have already been written, as well as a list of persons.

I’ll assume that I have a python list H containing people, and that Q(h) returns whether h is quixotic and P(h) returns whether h is pernicious.

```python
def S1(H) :  
    for h in H :  
        if Q(h) and not P(h) : return False  
    return True
```

As a variation, assume that P is a Python list of all pernicious people, and that Q is a Python list of all quixotic people. Write a Python function that takes P and Q as arguments, and returns True exactly when S1 is true.

I can check whether Q is a subset of P, which almost checks S1 (it doesn’t do anything for elements that are in neither P nor Q).
def subset(L1, L2):
    return False not in [x in L2 for x in L1]

subset(Q,P)

**Geography. Which, if any, of the following matches (S1):**

1. **The city of pernicious people is in the country of quixotic people.** No, we might then have some quixotic people who weren't pernicious, living outside the city.
2. **The city of quixotic people is in the country of pernicious people.** Yes, now only pernicious people are (by living in Quixotia, or whatever) quixotic.

**Does this image make any of the earlier questions easier for you, or at least quicker to answer?** Yes, so long as I make sure that I'm thinking of residents of the country of pernicious people and residents of the city of quixotic people (as opposed to citizens, where things get murkier).

**Familiar properties.** No solutions provided. This should be extremely similar to the preceding material.

**More Universal Quantification.** Write

(S3) **All ribald people are salacious**

in the form of (S1).

(S3') Only salacious people are ribald

2. **Some CSC108 logic review**

(1) **Is there a Python expression equivalent to**

   not (a and b)

   **that doesn't use and?**

   Yes

   (not a) or (not b)

(2) **Is there a Python expression equivalent to**

   not (a or b)

   **that doesn't use or?**

   Yes

   (not a) and (not b)