CSC110 Tutorial 2: Functions, Logic, and Autocorrecting with Predicates

Print this handout

This week in lecture, we reinforced and extended our understanding of how to write and test our Python functions, using two testing libraries called `doctest` and `pytest`. We then learned about formal logic as a tool for formally specifying boolean expressions, and applied what we learned to two new forms of Python code: *filtering comprehension* and *if statements*.

Now, let's review all of this on this tutorial!

Tip: to save the starter files to your computer, you can right-click on each link and select "Save Link As…" or "Download Linked File As…" (on Safari).

Exercise 1: Function Design Practice

Use the Function Design Recipe to design, implement, and test your work for each of the following problems. Complete your work in the starter file `tutorial2_ex1.py`, which you should save to your `csc110/tutorials/week02` folder.

1. Given a list of strings, return the average length of the strings, or return `-1.0` if the list is empty.
2. Given a set of strings and a non-negative integer, return a set containing the given strings whose length is less than or equal to the given integer.

   Hint: review the Course Notes Section 3.3 Filtering Collections for how to use a comprehension with a condition.

Practice testing your functions in two ways:

- First run your `tutorial2_ex1.py` file in the Python console and evaluate your doctest examples manually.
- Use the same code we saw in class for automatically running your doctest examples using Python's `doctest` module:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod(verbose=True)
```

Warning: due to a technical interaction between PyCharm and `doctest`, you cannot run doctest examples in the Python console. Instead, look for the green "Run" button appearing next to the `if __name__ == '__main__'` block, and click it and select **Run 'tutorial2_ex1'**.

Exercise 2: Propositional and Predicate Logic

Please complete the following questions to reinforce your understanding of logic from lecture.

1. Students often get stuck with the difference between the **AND** and implication operators when combined with the universal quantifier. In this exercise, you'll explore the difference between these two operators.

   Consider the following domain. Let M be a set of movies, and suppose we define the following predicates:

   - $SciFi(m)$: "m is a science fiction movie", where $m \in M$
   - $HighlyRated(m)$: "m is highly rated (has an average rating on IMDB of at least 8/10)", where $m \in M$.

   a. First, translate each of the following statements into English:

      1. $\forall m \in M,\ SciFi(m) \land HighlyRated(m)$
      2. $\forall m \in M,\ SciFi(m) \Rightarrow HighlyRated(m)$
      3. $\forall m \in M,\ HighlyRated(m) \Rightarrow SciFi(m)$

   b. In the space below, we've started a table that represents a possible state for a set M that contains four movies, with one row missing. Fill in the row so that all three of the above statements **True**.

Movie	Science fiction?	Score on IMDB
1	Yes	9.5
2	Yes	8.5
3	Yes	10.0
4		

   c. Fill in the row to make Statement 1 **False** and Statements 2 and 3 **True**.

Movie	Science fiction?	Score on IMDB
1	Yes	9.5
2	Yes	8.5
3	Yes	10.0
4		

   d. Fill in *two rows* to make all three of the above statements **False**.

Movie	Science fiction?	Score on IMDB
1	Yes	9.5
2	Yes	8.5
3	Yes	10.0
4		
5		

   e. Is it possible to complete the table with one row to make Statement 1 **True** and Statement 2 **False**? If so, do it. If not, explain why not.

Exercise 3: Debugging corner

Please download the starter file `tutorial2_ex3.py`, which contains two different functions. Each of these functions has a *small error* in its implementation, so that is behaves correctly on some inputs but not others.

For each function in this file:

1. Read the function description to understand what it's supposed to do, and then read the function implementation.
2. Fill in the two provided unit tests that call the function: one test that passes (i.e., the function is correct for the arguments you provide), and one test that fails or raises an error.

   You can write multiple test cases until you find one that passes and one that fails.
3. Explain, in English, what the error in the function implementation is.
4. Finally, fix the error.

Warning: you might be tempted to skip steps 2 and 3 and jump right to fixing the error, but please don't! Being able to demonstrate an error with a concrete failing test, and describe the error in English, are important communication skills for software engineers that we want you to develop this year.

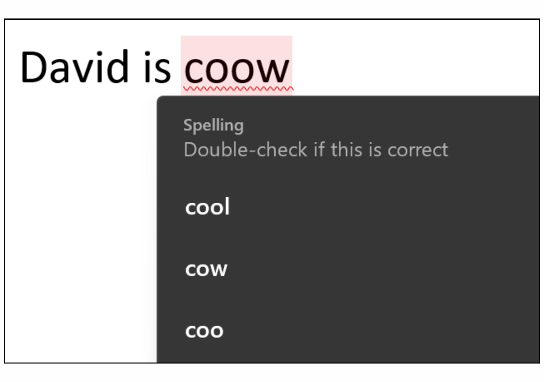
Exercise 4: Running PythonTA

This exercise is designed to help you get used to running PythonTA to check a Python file, which you'll need to do for both Prep 3 and Assignment 1. If you feel comfortable running PythonTA already because you've been using it for Assignment 1, you can skip this exercise. But if you haven't yet run PythonTA, we *strongly recommend* doing it now in tutorial so your TA can help answer any questions that come up!

To complete this exercise, download the starter file `tutorial2_ex4.py` and follow the instructions in the file.

Exercise 5: Predicates on strings and a simple autocorrect program

In this exercise, we'll combine everything we've learned about predicates, filtering collections, and if statements to write a small program that corrects simple spelling mistakes.



To work on this exercise, please download `tutorial2_ex5.py` and `tutorial2_words.py` into your tutorial folder.

1. First, let S be the set of all strings. We define the following term for strings:

   - Let $s_1, s_2 \in S$. We say that s_2 is a **suggestion** for s_1 when $|s_1| = |s_2|$ (that is, they have the same length), s_2 is an English word, and s_1 and s_2 differ in exactly one character.

     For example, if s_1 is the string `'tre'`, both `'are'` and `'the'` are suggestions for s_1. The string `'tree'` is *not* a suggestion for s_1 because it doesn't have the same length as s_1.

   Inside `tutorial2_ex5.py`, your first task is to complete the functions `num_differing_characters`, `is_suggestion`, and `get_suggestions`.

2. Next, implement `autocorrect_word`, which takes a string consisting of lowercase letters, and returns:

   - the string itself, if it is already a valid English word
   - the string itself, if is an invalid English word and it has no suggestions
   - the first suggestion for the string, if it isn't a valid English word and it has at least one suggestion (here "first" means "first alphabetically")

   Then, implement `autocorrect_words`, which takes a list of lowercase letter strings and returns a new list where each word has been autocorrected (by calling `autocorrect_words` on it).

3. Finally, put this all together in the function `autocorrect_text`, which takes a string of multiple words, and autocorrects each of them!

Taking this further

The small text autocorrect program you've written here is pretty cool, but is pretty limited. Here are some ways that you could think about extending this program:

- A broader range of words (including capitalized words)
- Allow a broader range of suggestions (multiple letters different, missing letters or extra letters, capitalized)
- Better ways of picking a suggestion to autocorrect (perhaps depending on the other words in the sentence!)

Feel free to experiment after this tutorial is over and see what you can come up with!

Additional exercises

1. *More Function Design practice*. Using the Function Design Recipe, write a function to solve each of the following problems.

   a. Given three floats, return whether any one of them is the sum of the other two.
   b. Given a list of strings and a non-negative integer n, return a new list that contains all of the strings with length n.

2. *Implication and its converse.*

   Using the same predicates as in Exercise 2, consider the following two statements.

   - Statement 1: $\forall m \in M,\ SciFi(m) \Rightarrow HighlyRated(m)$
   - Statement 2: $\forall m \in M,\ HighlyRated(m) \Rightarrow SciFi(m)$

   Now answer the following questions about these statements.

   a. First, translate each statement into English.

   b. Add *one* row to the table below that makes both statements **True**.

Movie	Science fiction?	Score on IMDB
1	Yes	9.5
2	Yes	8.5
3	Yes	10.0

   c. Add *one* row to the table below that makes Statement 1 **True** and Statement 2 **False**.

Movie	Science fiction?	Score on IMDB
1	Yes	9.5
2	Yes	8.5
3	Yes	10.0

   d. Add *one* row to the table below that makes Statement 1 **False** and Statement 2 **True**.

Movie	Science fiction?	Score on IMDB
1	Yes	9.5
2	Yes	8.5
3	Yes	10.0

3. Consider the following statement:

$$\Big[\big(\exists x \in U,\ P(x)\big) \land \big(\exists y \in U,\ Q(y)\big)\Big] \Rightarrow \Big[\exists z \in U,\ P(z) \land Q(z)\Big].$$

   Define a non-empty set U and predicates P and Q for which this statement is **False**.

   Hint: first translate this statement into English to help you understand what it is saying.

4. One of the important mechanical skills to be able to perform on boolean expressions is to negate them. Here are the rules governing how to simplify negations of expressions in predicate logic:

   - $\neg(\neg p)$ becomes p.
   - $\neg(p \lor q)$ becomes $\neg p \land \neg q$.
   - $\neg(p \land q)$ becomes $\neg p \lor \neg q$.
   - $\neg(p \Rightarrow q)$ becomes $p \land \neg q$.
   - $\neg(p \Leftrightarrow q)$ becomes $(p \land \neg q) \lor (\neg p \land q)$.
   - $\neg(\exists x \in S,\ P(x))$ becomes $\forall x \in S,\ \neg P(x)$.
   - $\neg(\forall x \in S,\ P(x))$ becomes $\exists x \in S,\ \neg P(x)$.

   Using these rules, simplify each of the following boolean expressions so that the negations are applied directly to predicates/propositional variables. Some expressions are written in predicate logic, while others are written in Python—in the Python expressions, you can simplify "not ==" to `!=`, "not <" to `>=`, etc. Note: this is a pretty mechanical exercise, but valuable to practice in both languages so that these transformations become "easy" for you.

   a. $\neg\Big((a \land b) \Leftrightarrow c\Big)$

   b. $\neg\Big(\forall x, y \in S,\ \exists z \in S,\ P(x, y) \land Q(x, z)\Big)$

   c. $\neg\Big(\big(\exists x \in S,\ P(x)\big) \Rightarrow \big(\exists y \in S,\ Q(y)\big)\Big)$

   d. `not ((a and b) or not c)`

   e. `not all([len(s) == 3 for s in my_strings])`

   f. `not (any([x < 0 for x in nums]) and any([x >= 0 for x in nums]))`