Homework 2

This homework will provide some practice with aspects of Python's types and data structures. Give each of these an earnest try in your Rmd before turning to stack overflow. You're also encouraged to post your question to Piazza - collaboration there on homeworks and labs is encouraged, including sharing approaches to solving questions. If you do use an external resource, be sure to cite it (include the url).

Types

Types or simple types are the primary forms that a single object in Python can take. They form the building blocks of more complex data structures.

1. What are the legal values that a Boolean type object can take in Python to indicate *true* and *false*? How about in R?
2. How would you check to see if a particular integer in Python is odd? (there are a few ways to skin this cat)
3. We've seen how you can use `and` and `or` to compose two Boolean type objects and output another Boolean. `xor` or "exclusive or" is a third option and though it exists in R as `xor()`, it doesn't in Python. Build up an expression that performs `xor` in Python using `and` and `or`.
4. Provide both the Python and R approaches of using `"hello"` and `" world"` to output
    1. `"hello world"`
    2. `"hellohellohello"`
    3. `"w"`
    4. `"olleh"`
5. We saw that in Python it is possible to concatenate and replicate strings using arithmetic operators. Do the update assignment operators also work on strings? Try it.
6. See Piazza for the note related to string comparison in Python.
    1. What does the output of `"Hi" < "lo"` and `"hi" < "Lo"` suggest about how the integers in the ASCII table associate with capital letters compares to that of lowercase letters? Check your answer with `ord()`.
    2. Place the following characters in order from least to greatest: `$`, `+`, `-`, `~`.
7. We discussed the important distinction between the notion of `equality` and `identity`. How do you check for these in R?

Data Structures

Data structures can also be called compound types or, frustratingly, just types (thus we still use `type()` to query them). The important distinction from the simple types is that they serve as structured containers for multiple simple objects. Each one is defined by its particular structure in terms of how indexing can be done, if at all, and if they are mutable, or capable of being changed without reassigning them.

1. Consider the list, `a = [1, 3, 5, ["a", "b"]]`.
    1. Replace the first element with the integer `99`.
    2. Extract the fourth element as a list using two approaches: positive and negative indexing.
    3. Extract `"a"` as a list with one element, a string.
    4. Extract from the list the elements and sub-element needed to form a new list: `[1, 3, "b"]`.
    5. In one line of code, reverse the order of the elements and subelements of the list so that the result is `[["b", "a"], 5, 3, 1]`.
2. What happens if you try to use negative indexing on a vector in R? Please show an example.
3. What are the closest R analogs to each of the four basic data structures in Python? Are they perfect matches? If not, how do they differ? You can get `reticulate`'s opinion on this by bringing different data structures back and forth between R and Python using `py$` and `r.`
4. The central data structure in R is the dataframe. Construct an analog in Python to this snippet of the `mtcars` dataframe in R (ignore the row names): `mtcars[1:3, 1:3]`.
5. Dictionaries look an awful lot like sets but with a key associated with each value in the set. Do set operations work on dictionaries? Try it out.

Farther afield

1. Why does Python use zero-based indexing? Propose a possible rationale of your own, then look online to see what other people think.
2. What is a factor in R? A type? A data structure?
3. What are the type heterogenous data structures and type homogeneous structure in R? Why do you think R in particular has so many type homogeneous structures? (feel free to speculate)