Problem Set IV: ASCII & Indexing

CPE 101: Fundamentals of Computer Science
WInter 2019 - Cal Poly, San Luis Obispo

Purpose

To attain more experience using lists and indexing and to begin to understand how characters - and thus, strings - are represented by a machine.

Description

ASCII Character Set

ASCII characters are represented in a machine as a single byte, which is a sequence of 8 bits. Each bit (or "binary digit") has a value of either 0 or 1. Thus, since each bit can hold one of two values and there are 8 such values in a byte, the number of possible values a byte can store is 2^8 or 256. As the English language comprises far fewer symbols than 256 (including numbers and punctuation), it is possible to represent each character in English in a byte. Thus, each character has a numerical representation, as can be referenced by entering `man ascii` in a terminal.

0	1	0	0	0	0	0	1
7	6	5	4	3	2	1	0

The bits displayed above represent the number 65, which, in ASCII, is the character "A". This number can be found by adding $2^0 + 2^6$ which equals 1 + 64 or 65.

Polynomial Arithmetic

You will implement functions that perform basic arithmetic on polynomials, such that each polynomial will be represented as a list. The values in the list will represent the coefficients of the terms whereas the indices will represent the exponents for the terms.

Thus, the polynomial $2x^2 + 3x + 5$ will be represented by the following list:

```
poly = []
poly.append(5)
poly.append(3)
poly.append(2)
```

This list can also be created directly as follows:

```
poly = [5, 3, 2]
```

Notice that the term with exponent 0 is first in the list while the term with exponent 2 is last; thus, the terms in the list are in reverse order of how they are typically written in mathematics so that an element's index represents that term's exponent. You may think this mapping of a polynomial to a list is a bit odd. In fact, attributing meaning to indices of a list (and not just the values within the list) is an important skill that allows a list to be used as more than just a substitution for a bunch of variables.

Implementation

You are required to write at least 3 tests (assert statements) for each of the functions below. Since we are emphasizing test-driven development, you should read the descriptions and write tests before implementing each function.

Do not refer to an ASCII table to complete this assignment. No ASCII numbers should be used in your implementation - instead, use letters passed to the `ord` function.

Character Manipulation

`is_lower(char)`

Return `True` if the given character is lowercase (assuming only the English alphabet) and `False` otherwise. Do not use the `str.islower` function.

`char_rot_13(char)`

Return the ROT-13 encoding of the given character, which is a simple Caesar-cypher encryption that replaces each character of the English alphabet with the character 13 places forward or backward along the alphabet (e.g. "a" becomes "n", "b" becomes "o", "N" becomes "A", etc.). This only works on the characters in the alphabet. All other characters are left unchanged. Moreover, the rotation is only within the characters of the same case (i.e. a lowercase letter always rotates to a lower case letter and the same for uppercase letters). An encoded character can be decoded by applying the rotation a second time. This function may use the `str.isalpha`, `str.islower`, and `str.isupper` functions.

String Manipulation

`str_rot_13(my_str)`

Return the ROT-13 encoding of the given multi-character string. This function must make calls to `char_rot_13` rather than duplicating its implementation.

`str_translate(my_str, old, new)`

Return a new string where each occurrence of string `old` is replaced with string `new` and all other characters are left unchanged. Do not use any built-in string functions with similar behavior, such as `str.replace`.

Example: `str_translate("abcdcba", "a", "x")` returns `"xbcdcbx"`

Polynomial Operations

Though the use of loops would allow one to generalize these functions, in the interest of practicing indexing you **cannot use any loops** for these problems.

`poly_add(poly1, poly2)`

Return a list representing the sum of the given polynomials, each of which has degree two (is a 3-integer list). This function must return a new list - do not modify the contents of the input lists.

`poly_mul(poly1, poly2)`

Return a list representing the product of the given polynomials, each of which has degree two (is a 3-integer list). This function must return a new list - do not modify the contents of the input lists.

Polynomial multiplication is not a simple multiplication of values at the same index; instead, think of the distributive law (of which the FOIL method is a simple case). The polynomial resulting from a multiplication will, in general, be of degree greater than the arguments. In this case, the result can be of at most degree four, so the returned list may be larger than that of the given polynomials.

Submission

Zip all the files into one zip file and submit it to Polylearn.