

In this project you will write a program to construct error correcting codes. Constructing a *good* error correcting code is *hard*, so our focus will not be on trying to create the best codes, but rather on using some of the basic counting principals we have learned (permutations, combinations, etc.) to do some small brute-force searches to create codes. The codes we create will be binary codes (meaning that we will only be dealing with data that is already a string of bits).

So that we are all solving the same problem, we are going to use the following definitions (regardless of what definitions you encounter on Wikipedia, use these for this project):

1. **Hamming Distance:** The hamming distance between two equal length binary strings is the number of bits that are different.
2. **Correction Tolerance:** The maximum hamming distance at which errors can be corrected. For example, a code which can always correct single bit errors, but can't correct a double bit error, has a correction tolerance of 1.
3. **Data Capacity:** The number of data bits which the code encodes.
4. **Code Size:** The number of bits used to represent the codewords.
5. **Code Efficiency:** The ratio of the data capacity to the code size.

For example, an error correcting code which encodes three bits of data into five bits codewords and can correct single bit errors has a data capacity of 3, a code size of 5, a code efficiency of $3/5$, and a correction tolerance of 1.

For your project, you must implement a function: `create_code(data_cap,tol)` which returns the code as a list of two Python dictionary objects: `[enc, dec]` which define a code with data capacity `data_cap` and correction tolerance `tol`. The `enc` object must take data strings as keys and return the appropriate codeword. The `dec` object must take `any codeword` as a key and return the data sting that is represented by the codeword. Note that this means the `dec` object should return the correct result for codewords that are modified within a hamming distance of `tol`.