Strings Cheat Sheet

**word[ pos]**
- Return the character at position pos.
- Positions are counted starting at 0 for the first character and going up, or starting at -1 for the last character and going down.

**word[ start:end], word[ start:end:step]**
- Return a string of characters, starting with the character at start, “counting” by step, and going to, but not including, the character at end.
- If step and the final colon are omitted (first form), then count by 1.
- If start is omitted: start with the first character, if step is positive; and start with the last character, if step is negative.
- If end is omitted: include the last character, if step is positive; and include the first character, if step is negative.

**word.index(sub), word.index(sub, pos)**
**word.find(sub), word.find(sub, pos)**
- If pos is not present (first form), return the smallest index in word where substring sub is found.
- If pos is present (second form), return the smallest index greater than or equal to pos where substring sub is found in word.
- If sub is not found: index produces an error, whereas find returns -1.

**word.replace(old, new)**
- Return a copy of word with all occurrences of substring old replaced by string new.

**len(word)**
- Return the length of word.

**word1 + word2**
- Return the string of characters made by concatenating a copy of word1 and a copy of word2 to form one string.

**word.lower()**
- Return a new string produced from word by converting all alphabet characters to lower case.

**word.upper()**
- Return a new string produced from word by converting all alphabet characters to upper case.
word1 < word2, word1 <= word2, word1 > word2, word1 >= word2
  • Compare characters in word1 and word2, starting from position 0; return True or False based on the relationship between the Unicode values of the characters at the first position where word1 and word2 differ.

word1 == word2, word1 != word2
  • Compare words for equality (==) or inequality (!=); return True or False.

word1 in word2, word1 not in word2
  • Check if word1 appears in word2 as a substring; the first expression returns True if it does, and False if it does not; the second expression returns False if it does, and True if it does not.

for c in word:
    suite

Repeatedly execute suite for each character in word; before each iteration, assign c the next character in word.