15-112, Week 3, Strings Practice

See how many of these problems you and a partner can solve during today's lab.

For each problem, you write the requested function *and* testcases for it. See `week01/example.py` in the demo code repository for an example of organizing this.

Rules for every problem below: no lists, no recursion. You have strings, loops, conditionals, and arithmetic — that's it.

---

1. collapseWhitespace(s)

Write `collapseWhitespace(s)`, which returns `s` with every run of consecutive whitespace collapsed to a single space. Don't use `s.replace()`. For example, `collapseWhitespace("a\t\t\tb\n\nc")` returns `"a b c"`.

```
assert(collapseWhitespace("a\nb") == "a b")
assert(collapseWhitespace("a\n   \t    b") == "a b")
assert(collapseWhitespace("a\n   \t    b  \n\n  \t\t\t c   ") == "a b c ")
```

---

2. replace(s1, s2, s3)

Write `replace(s1, s2, s3)`, your own version of `s1.replace(s2, s3)` (replacing every non-overlapping occurrence of `s2` in `s1` with `s3`) — without calling `s.replace()`. For example, `replace("banana", "an", "X")` returns `"bXXa"`.

---

3. interleave(s1, s2)

Write `interleave(s1, s2)`, which interleaves the characters of two strings starting with `s1`'s first character, appending any leftover characters from the longer string at the end. For example, `interleave('pto', 'yhn')` returns `"python"`, and `interleave('a#', 'cD!f2')` returns `"ac#D!f2"`.

---

4. wordWrap(text, width)

Write `wordWrap(text, width)`, which line-wraps `text` (lowercase letters and spaces only) at `width` characters, then strips leading/trailing spaces from each line and converts any remaining spaces to dashes. So `wordWrap("abc", 2)` returns a 2-line string: `"ab"` then `"c"`.

```
assert(wordWrap("abcdefghij", 4) == "abcd\nefgh\nij")
assert(wordWrap("a b c de fg", 4) == "a-b\nc-de\nfg")
```

---

5. sameChars(s1, s2)

Write `sameChars(s1, s2)`, which returns `True` if the two strings use exactly the same set of characters (regardless of count or order), and `False` otherwise. Case-sensitive: `"ABC"` and `"abc"` don't match. Returns `False` if either argument isn't a string, but `True` if both are empty (why?).

For example, `sameChars("aabbc", "abc")` returns `True`, while `sameChars("aabbc", "abcd")` returns `False`.

---

6. hasBalancedParentheses(s)

Write `hasBalancedParentheses(s)`, which returns `True` if the parentheses in `s` are balanced (ignoring all other characters), and `False` otherwise. For example, `"( ( ( ) ( ) ) ( ) )"` is balanced, but `"( ) )"` and `"( ) ) ("` are not. *Hint: track how many left parentheses are currently unmatched.*

---

7. encrypt(plaintext, password) and decrypt(ciphertext, password)

Background: this is a simple password-shift cipher. Take a plaintext message, drop non-letters, and uppercase the rest — e.g. `"Go team!"` becomes `"GOTEAM"`. A password (all lowercase, e.g. `"azby"`) gives a repeating sequence of shifts, where `a`=0 ... `z`=25 (so `"azby"` → shifts `0, 25, 1, 24`, repeating). Shift each plaintext letter by the corresponding password shift, with wraparound. So `"GOTEAM"` shifted by `"azby"` (repeated) becomes `"GNUCAL"`.

encrypt: write `encrypt(plaintext, password)`, implementing the process above. If `password` isn't all lowercase, return `"password must be all lowercase"`. *Hint: `ord(c)` and `chr(v)` convert between a character and its Unicode value.*

decrypt: write `decrypt(ciphertext, password)`, which reverses `encrypt`. For example, `decrypt("GNUCAL", "azby")` returns `"GOTEAM"`.