Strings IV

- Write a function called count_dups that counts the number of back-to-back duplicated characters in a string.
  - count_dups("balloon") returns 2.

- Count duplicates:

```
def count_dups(s):
    total = 0
    for pos in range(0, len(s), 1):
        if <test s[pos] for something>:
            total = total + 1
    return total
```

Is s[pos] the same character as the character immediately to the right?

- Count duplicates:

```
def count_dups(s):
    total = 0
    for pos in range(0, len(s), 1):
        if s[pos] == s[pos+1]:
            total = total + 1
    return total
```

Is s[pos] the same character as the character immediately to the right?

- Count duplicates:

```
def count_dups(s):
    total = 0
    for pos in range(0, len(s)-1, 1):
        if s[pos] == s[pos+1]:
            total = total + 1
    return total
```

Is s[pos] the same character as the character immediately to the right?

`s.startswith(t)`	True if the string s begins with the string t.
`s.endswith(t)`	True if the string s ends with the string t.

`s.find(t)`	Returns the lowest index at which substring t is found inside s.
`s.find(t, p)`	Same as above, but starts searching at position p.
`s.replace(t, t2)`	Returns a copy of s with all occurrences of t replaced by t2.

`s.upper()`	Returns a copy of s with all letters converted to uppercase.
`s.lower()`	Returns a copy of s with all letters converted to lowercase.

Three common string computations

- Count the number of times something happens in a string.
- Filter a string to keep only the characters that satisfy some condition.
- Transform a string into a new string by changing each character in some fashion.

Counting

look at each character:

does this character match a pattern?

if yes, increment total

```
total = 0
for pos in range(0, len(s), 1):
  if <test s[pos] for something>:
    total = total + 1
```

Filtering

look at each character:

does this character match a pattern?

if yes, attach the character to the answer

```
answer = ""
for pos in range(0, len(s), 1):
  if <test s[pos] for something>
    answer = answer + s[pos]
```

Transforming

look at each character:

turn this character into a new character and attach it to the answer

Transforming

look at each character:

turn this character into a new character and attach it to the answer

```
answer = ""
for pos in range(0, len(s), 1):
    newchar = <do something to s[pos]>
    answer = answer + newchar
```

Transforming

Turn every character into a hyphen:

```
answer = ""
for pos in range(0, len(s), 1):
   newchar = "-"
   answer = answer + newchar
```

Transforming

Common to use an if statement inside a transform:

```
answer = ""
for pos in range(0, len(s), 1):
  if <something>:
    answer = answer + <something>
  else:
    answer = answer + <something else>
```

Transforming

Switch the case of all letters (lower <---> upper)

```
answer = ""
for pos in range(0, len(s), 1):
   if <something>:
      answer = answer + <something>
   else:
      answer = answer + <something else>
```

Transforming

Transforming is often combined with filtering.

How can we change our function so uppercase/ lowercase are switched, and everything else is removed?

- Write a function called change_nums that increments all numbers in a string by one:
  - Example: change_nums("a1b2") returns "a2b3"
- Write a function called encode that takes a string and encodes it using the simple cipher A=1, B=2, C=3, and so on.
- Example: encode("abc") returns "1-2-3".
- Hint: use a variable letters = "abcdefgh…" and the find function.
  - What is letters.find("a")? letters.find("b")?
- Challenge (hard): write a decode function.