Lambda and reduce

From composing program textbook.

```
def compose(f,g):
    return lambda x: f(g(x))
```

A function that takes x and returns f(g(x))

Ex s=lamba x: x*x

⇒ s(12) = 144

```
def reduce(reduce_fn, s, initial):
    reduced = initial
    for x in s:
        reduced = reduce_fn(reduced, x)
    return reduced
```

reduce_fn: Call back f'n; s: sequence; initial: starting from

Functions / Methods used in EPI

- lambda
- reduce → functools
- hexdigits
- ord
- chr
- zip
- map
- list
- reversed
- all
- any
- enumerate

reduce → Takes a list and perform the input f'n from left to right.

from Itertools module
- groupby
- accumulate
- product
- combinations

7/2/22

6.2 Correction

Input = "615", $b_1 = 7$, $b_2 = 13$.

Then the # value expressed in decimal. is 306

⇒ 306 % 13 = 7 then continue to 306 // 13 = 23

⇒ 23 % 13 = 10 = "A" then 23 // 13 = 1

⇒ 1 % 13 = 1 and 1 // 13 = 0

⇒ 1, 10, 7 ⇒ "1A7"

Evaluation

- Need to learn functools → reduce, lambda

EPI 6.2 Soln

```
def convert_base(num_as_str:str, b1:int, b2:int)->str:
    def construct_from_base(num_as_int, base):
        if num_as_int == 0:
            return ''
        else:       # recursive step
            return(construct_from_base(num_as_int // base
            base) + string.hexdigits[num_as_int % base]
            .upper())

    is_neg = num_as_str[0] == '-'
    num_as_int = functools.reduce(
        lambda x,c : x*b1 + string.hexdigits
        .index(c.lower()),
        num_as_str[is_neg:], 0)

    return ('-' if is_neg else '')
        + ('0' if num_as_int == 0 else
        construct_from_base(num_as_int, b2))
```

func(num // base)

remainder

2/7

construct_from_base (306 // 13, 13)
+ '7'

= '7'

repeat step because recursion

cstr_from_base (23 // 13, 13) + 'A' = 'A7'

repeat step

—— (1 // 13, 13) + '1' = '1A7'

6.2

$b_1, b_2 \in \mathbb{Z}$

I/O : strings, b_1 and b_2 / string representing an $\mathbb{Z}$ in base b_2.

Constraints:

$2 \leq b_1$,

$b_2 \leq 16$

"A" represents 10, "B" rep. 11

Ex Input "615" then $b_1 = 7$, $b_2 = 13$

Output "1A7"

$6 \times 7^2 + 1 \times 7 + 5 = 1 \times 13^2 + 10 \times 13 + 7$

ignore?

7min & have no idea

What if the ~~multiplication~~ product doesn't exist? → edge case? return 0?

4/7

Not understanding the problem.

Ex

```
   49          169
    6          130
  294            7
    7
    5
```

615 → 306 + 306 = 612

Should I 615 // 2 ?

↳ = 307

↳ If even -1 ?

↳ 306

↳ And find this?

The plan

So divide by 2, if odd minus one.

Then find value where the b_1 and b_2 can represent the base.

Then return $0 \leq a_1 < b$ that denotes the halved input string w/ base 2 (b_2)

Can I use pythonic method int()? 5/7

halved = int(5) // 2

IF halved is odd,
THEN halved - 1

In the example,

$(6 \times 7^2) + (1 \times 7) + 5 = (1 \times 13^2) + (10 \times 13) + 7$

Should I use this?

where the tenth digit to the return value is b_1?

IF that is true, then we should halve the input and check to see if the tenth digit is b_1.

If it does exist, continue.

6/7

So from the example.

617 ⇒ 617//2 = 307

↳ ⇒ 307 − 1 = 306

Now, 306 − 7 = 299

While 299 ⩾ 0

$299 - 13^2 = 130$

---

30 min quit

Can't figure it out.

6.4 Corrections

IF no 'a', we can implement func. w/ one forward iteration by skipping 'b's and copying over the other characters.

```
write_index = 0, a_count = 0
for i in range(size):
    if s[i] ≠ 'b':
        s[write_index] = s[i]
        write_index += 1

    if s[i] == 'a':
        a_count += 1.
```

---

we have that [a, b, a, c, ...] and we will return [a, a, c, ...]

0 1 2 3

{a, b, a, c, ␣, ␣, ␣, ...}

a_count = 1

s[w_i = 0] = s[i = 0] = 'a'

w_i = 1

⇒ [a, ...]

Nothing happens

a_count = 2

s[w_i = 1] = s[i = 2] = 'a'

w_i = 2

⇒ [a, a, ...]

s[w_i = 2] = s[i = 3] = 'c'

w_i = 3

⇒ [a, a, c, ...]

IF there is no 'b's, compute the final length of the resulting string which is length of array plus number of 'a's.
And write the result from backwards.

From the last example, we have

[a, a, c, ...]

w_i = 3
a_c = 2

---

curr_index = w_i - 1 = 2
w_i += 2 - 1 = 4
final_size = w_i + 1 = 5

```
while curr_index >= 0:
    if [curr_index] == 'a':
        s[w_i-1 : w_i+1] = 'dd'
        w_i -= 2
    else:
        s[w_i] = s[curr_index]
        w_i -= 1
    curr_index -= 1

return final_size
```

0 1 2 3 4

[a, a, c, ـ, ـ, ـ]

↑

s[w_i = 4] = s[curr_index // 2] = c

w_i = 3

↑

s[w_i - 1 = 3 - 1 = 2 : w_i + 1 = 3 + 1 = 4]

= 'dd' (which means s[2] = d, s[3] = d)

w_i -= 2 ⇔ w_i = 1

⇒ [a, a, d, d, c]

↑

s[w_i - 1 = 1 - 1 = 0 : w_i + 1 = 1 + 1 = 2]

= s[0:2] = 'dd' ⇔ s[0] = 'd', s[1] = 'd'

w_i = -1

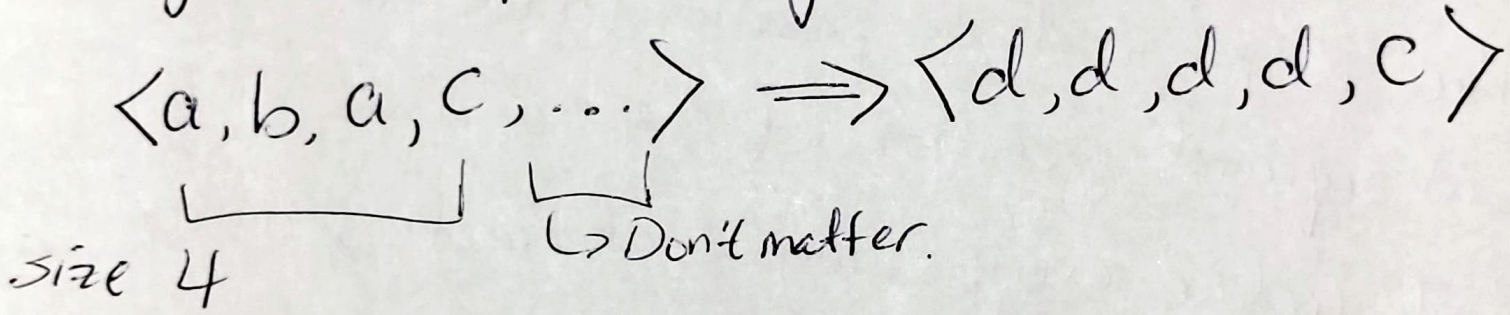
⇒ [d, d, d, d, c]

6.4

Input: Array of characters and size of the entries.

Output: replace 'a' w/ two 'd' and del. 'b'

The size is how many times we iterate through the input array.



Constraints: The ~~input size will~~ is enough space in array to hold final result.

A Naive sol'n is to iterate through the input array up to the size input and use ~~the loop~~ conditionals to check if 'a' or 'b'. And insert or delete from there. This will take O(n) time and space where n is the length of the array.

4/8

- Since the items after the input size doesn't matter I can replace 'b' to end of the array (or anyplace after size).
- For 'a''s maybe I can replace the item in that position and the subsequent position (so i and i+1) and replace every single characters after?
  ↳ Maybe bad idea.

Hint says to do multiple passes
↳ Why? ↳ Recursive function?
↳ $O(n^2)$

- Shouldn't there be a sol'n where I can just append to a new list?

15min

7/8

Naive Sol'n

```
def func(A: List[str], s: int) -> List[str]:
    result = []
    for i in range(s):
        if A[i] == 'a':
            result.append('d')
            result.append('d')
        if A[i] == 'b':
            continue
        else:
            result.append(A[i])

    return result.
```

O(n) time and space ~~Maybe~~

↳ (20 min)

8/8

6.6 Evaluation

- Had similar idea for the loop to reverse each word but couldn't think to reverse the whole string first.

- The solution was to reverse the whole string, then iterate through the string to check for white spaces.
- For every whitespace, reverse the substring using the helper f'n.
- The trick here was to create a helper f'n to reverse a string w/ set parameters.

- Reversing the whole string will get their correct relative positions.
- Remember to split each characters first (the book doesn't do this for some reason)

6.6

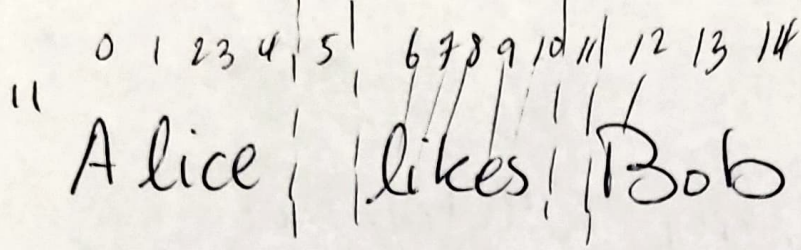
Input: String containing whitespace
Output: String reversed by whitespace.

- The Naive sol'n will be to split the input string by space and loop and append to a new list in reverse order.
- This will take O(n) time and space.
- There is probably a sol'n w/ O(1) space.
  ↳ where we swap positions.

(*) Splitting the string will take additional space. ~~since strings~~ since we append to a new list.

↳ Furthermore, we have to join the string back together.

⟹ Therefore, it's better to iterate by character by char.



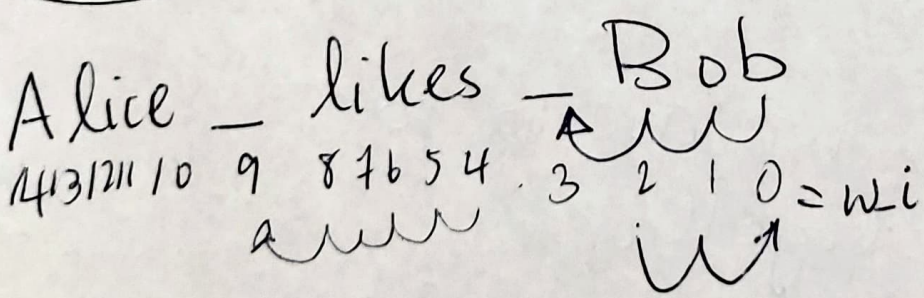
First, we can count the length of the string.

EX The length of s is 15.

Note: I'm trying to avoid split string by whitespace

Then we can switch the chacters from backwards while checking for whitespace.

we can start the iteration from backwards and swap characters in the front.



Bob—
0 1 2 3

```
w_i = 0
for i in ~~range~~( reversed(range(s)):
    if s[i] == '_':
        for j in range(w_i, i+1):
            s[j] = s[w_i]
        w_i += i + 1

return s
```

30min gave up