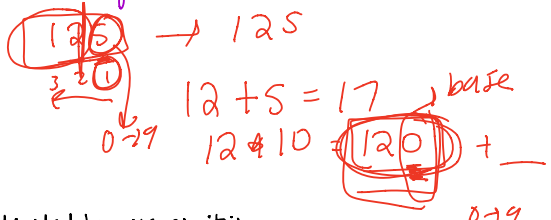
Recursion

① a) positionizer

Input → non-negative integer n
returns → non-negative integer
main idea → for each digit d of n,
if d is equal to its position,
leave d the same
else,
change d to be remainder of d divided by its position

④ position of rightmost digit of # is 1
position of digit increases R→L



eg.) from doctest #1

```
>>positionizer(12)
```

n = 1 2
pos 2 pos 1

d = 2 → not same as position #
position = 1
so, new d = 2/1 = 2, R = 0

output = ␣0
d = 1 → not same as position #
position = 2
so 1/2 = 0, R = 1

output = 10

(10)

```
def positionizer(n):
    def helper(n, pos):
        if _____:
            return _____
        rest = _____
        if n%10 == pos:
            return rest + _____
        else:
            return rest + _____
    return helper(_____, _____)
```

higher order functions are useful/helpful if we have recursion but need to keep track of more things than just the parameter(s) we have

base case

keep the rest of the # the same as you change only one digit

remember when we have a choice, R→L traversal easier w/ we have more tools (specified in this & the two)

"easy" to fill in one of the blanks given that we know what n should be & what first pos would be

SOLUTION FOR THIS QUESTION BELOW!!

⇓

```
def positionizer(n):
    def helper(n, pos):
        if n == 0:
            return 0
        rest = helper(n//10, pos + 1) * 10
        if n%10 == pos:
            return rest + (n%10)
        else:
            return rest + (n%10)*10**pos
    return helper(n, 1)
```

stopping condition, continually calling helper(n//10, pos) and want to stop when n//10 = 0 cuz that means we've gone through all digits

*10 is helpful tool so that we can do rest + ____ below

this is case when we don't need to change d

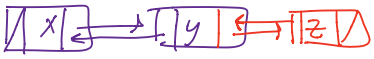
need to change d

start w/ same n & position 1

d

OOP (LL)

② a) DLList → prev (previous link) → value (current) → next (next link)




```
class DLList:
    empty = None
    def __init__(self, value, next = empty, prev = empty):
        ____
        ____
        ____
```

(#) class example(test):

this is same as when we did Link.empty to signify the end of the linked list

pretty much same as Link but w/ extra parameter

self is like implicit reference to myself (object which calls this) but need to always do self. to get attributes of given object

initialize everything (params)

```
def add_last(self, value):
    pointer = self
    while ____:
        ____
    ____ = DLList(____)
```

add value to the end of the current linked list

store the first link (self) into the variable named pointer

need to somehow iterate to the end of the current DLList

use this to construct the last DLList object for the list

SOLUTION BELOW !!

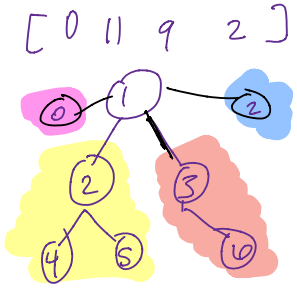
```
class DLList:
    empty = None
    def __init__(self, value, next=empty, prev=empty):
        self.value = value
        self.next = next
        self.prev = prev

    def add_last(self, value):
        pointer = self
        while pointer.next != DLList.empty:
            pointer = pointer.next
        pointer.next = DLList(value, DLList.empty, pointer)
```

Trees

① a) equally_weighted(t)

total weight → sum of all labels in nodes (function provided)

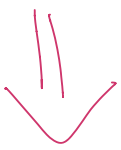
equally_weighted → total weight of branches are equal
(leaf is assumed to be equally weighted)




```
def equally_weighted(t):
    ____ = [ ____ ]          # somehow get the weights of all the branches in one list
    for ____ in ____:        # iterate through weights of all the branches
        if ____:             # somehow check if branch weight is same as all others
            return ____
    return ____
```

need to return either true or false

ANSWER BELOW!!



```
def equally_weighted(t):
    all_weights = [total_weight(b) for b in t.branches]
    for weight in all_weights[1:]:
        if weight != all_weights[0]:
            return False
    return True
```

[x:y:z] → inclusive start, exclusive end, increment

Generators

① a) n_apply

n_apply(f, n, x)
↳ apply function f to x, n times
eg.) for n=3, f(f(f(x)))

f(f(f(x1))) x2
f(f(x2))
f(x3) x4
x: f(x) → x = f(x)
x = f(f(x))

```
def n_apply(f, n, x):
    for _____:
        x = _____
    return _____
```

iterate n times updating x everytime

→ output of last f(x) where x = f(f(x)) ← input if n=3

ANSWER BELOW

```
def n_apply(f, n, x):
    for i in range(n):
        x = f(x)
    return x
```

b) list_gen

list_gen(lst, f)

↳ for element at index i of lst, lst_gen applies f to element i times & yields that value lst[i] times

① generators use yield / yield from

② can continue to have code after yields

```
def list_gen(lst, f):
    for _____:
        yield from [_____]
```

need to iterate through entire list

need an iterable list so all the values are here

need to make a call to n_apply w/ proper parameters

ANSWER BELOW

```
def list_gen(lst, f):
    for i in range(len(lst)):
        yield from [n_apply(f, i, lst[i]) for j in range(lst[i])]
```