Task 1:
Assuming `equals` and `if` are already defined
Define `let` for cleaner syntax
$[let\ n = t_2\ in\ t_3] = [(\lambda n.[t_3])[t_2])]$
$let\ Y = \lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))\ in$
$let\ pair = \lambda x.\lambda y.\lambda z.z\ x\ y\ in$
$let\ first = \lambda p.p(\lambda x.\lambda y.x)\ in$
Instead of representing `[a,b,c]` as `pair a (pair b c)`, which doesn't give a way
to indicate list end, we can represent the list like this:
`pair false (pair a pair (false pair (b pair (false pair (c pair (true true))))))`
where `pair true true` is used to represent an empty list and
`pair false list` represents a non-empty list `list`
Thus, we can define nil like this:
$let\ null = pair\ true\ true\ in$
$let\ isNull = first\ in$ # list is null if first element is true
$let\ cons = \lambda head.\lambda rest.pair\ false\ (pair\ head\ rest)\ in$
$let\ head = \lambda list.first\ (second\ list)\ in$ # second.fist is the head
$let\ rest = \lambda list.second\ (second\ list)\ in$ # second.second is the rest of the list

Task 2:
Map over a `list` using a `callback`
$let\ map = Y\ (\lambda map'.\lambda callback.\lambda list.$
$\quad if\ (isNull\ list)$
$\quad\quad null$
$\quad\quad (cons\ (callback\ (head\ list))\ (map'\ callback\ (rest\ list)))$
$)\ in$

Task 3:
Return `value` if it is present in the `list`. Otherwise, return `null`
$let\ search = Y\ (\lambda search'.\lambda value.\lambda list.$
$\quad if\ (isNull\ list)$
$\quad\quad null$
$\quad\quad (if\ (equal\ (head\ list)\ value)$
$\quad\quad\quad value$
$\quad\quad\quad (search'\ value\ (rest\ list))$
$\quad\quad)$
$)\ in$

Helpers
$let\ inc = \lambda value.add\ value\ 1\ in$
$let\ fin = null\ in$

$let\ list\ =\ cons\ 1\ (cons\ (2\ (cons\ 3\ null)))\ \ in$

Task 4:
$let\ task\_4\ =\ map\ inc\ list\ in$

Task 5:
$let\ task\_5\ =\ search\ 2\ list\ in$

Task 6:
$let\ task\_6\ =\ search\ 4\ list\ in$

fin