Homework 3

Tejas Kamtam

305749402

CS 131 - Fall 2023

Problem 1

Part a

```
ll_contains :: LinkedList → Integer → Bool
ll_contains EmptyList _ = False
ll_contains (ListNode x xs) y
    | x == y = True
    | otherwise = ll_contains xs y
```

Part b

```
ll_insert :: (Eq LinkedList) ⇒ LinkedList → Integer →
Integer → LinkedList
```

The function should take the linked list, position, and value as input. The position must be a whole number so it must be an Integer (or Int). The type definition of a ListNode specifies the data type must be an Integer. Because all data/variables is immutable and all functions must return something, and the insert operation recreates nodes in path up to the root (head), we should return this new LinkedList with the value inserted.
We also add the `Eq LinkedList` constraint to allow comparing LinkedLists.

Part c

```
ll_insert :: (Eq LinkedList) ⇒ LinkedList → Integer →
Integer → LinkedList
ll_insert (ListNode x xs) pos y
  | pos ⩽ 0 = ListNode y (ListNode x xs)
  | xs == EmptyList = ListNode x (ListNode y EmptyList)
  | otherwise = ListNode x (ll_insert xs (pos-1) y)
```

Problem 2

Part a

Python passes parameters by object reference (i.e., pointers), so when `num` is passed to `quintuple()`, the function points num to a new object with the value `num*5`, BUT when we go back to print the original passed parameter's object that it is pointing to (since all variables are also object references), it remains unchanged since the pointer was made to change the value of a different object within the function.
On the other hand, changing the attribute of an object that the pointer is still pointing to does not reassign the pointer, so the original object is truly changed, thus we get `15` as the output.

Part b

i

The program would print:

```
joke6
joke3
joke4
```

ii

The program would output an error since we haven't imported `copy`, but assuming we did, the program would print:

```
joke6
joke2
```

Problem 3

Python is dynamically typed, so it will not check whether or not you CAN run `len` on the object, but when executing/interpreting it will try to call that object's `__len__()` method, which is not defined for integers, so it throws an error.

Problem 5

Part a

```
def largest_sum(nums, k):
  if k < 0 or k > len(nums):
    raise ValueError
  elif k == 0:
    return 0

  max_sum = None
  for i in range(len(nums)-k+1):
    sum = 0
    for num in nums[i:i+k]:
      sum += num
    if max_sum == None or sum > max_sum:
      max_sum = sum
  return max_sum
```

Problem 6

Part a

```
class Event:
  def __init__(self, start, end):
    if start > end: raise ValueError
```

```
    self.start_time = start
    self.end_time = end
```

Part b

```
class Calendar:
  def __init__(self):
    self.__events = []
  def get_events(self):
    return self.__events
  def add_event(self, event):
    if type(event) ≠ Event: raise TypeError()
    self.__events.append(event)
```

Part c

We get this error because although we inherit from the Calendar class, we do not call the Calendar constructor, so the `__events` attribute is never initialized. We can solve this many ways, 2 of which are including either of the following lines in the `__init__()` method of the `AdventCalendar` class:

Method 1

```
super().__init__()
```

or in earlier versions of Python:

```
super(AdventCalendar, self).__init__()
```

Method 2

```
Calendar.__init__(self)
```