Contents

1 Basic Test Results

```
Starting tests...
Mon Nov 29 07:53:51 IST 2021
5951ab57e978cf833b0f402ad0f112c5d50ced8a  -


Archive:  /tmp/bodek.Z_cuBu/intro2cs1/ex7/eyalmutzary/presubmission/submission
  inflating: src/ex7.py


Running mypy
Success: no issues found in 2 source files
Finished running mypy

Running presubmit code tests...
9 passed tests out of 9 in test set named 'ex7'.
result_code    ex7    9    1
Done running presubmit code tests

Finished running the presubmit tests

Additional notes:

The presubmit tests do not check if you used functions or operators you are not
supposed to use.

Make sure to thoroughly test your code.

```

2 ex7.py

```
1  ##################################################################
2  # FILE : ex7.py
3  # WRITER : eyal , eyalmutzary , 206910432
4  # EXERCISE : intro2cs ex7 2021
5  # DESCRIPTION: Recursions!
6  # STUDENTS I DISCUSSED THE EXERCISE WITH:
7  # WEB PAGES I USED:
8  # NOTES: ...
9  ##################################################################
10 from typing import Any, List
11 from ex7_helper import *
12
13
14 # --------- Part 1 ---------
15
16
17 def mult(x: float, y: int) -> float:
18     """
19         multiplies x * y
20         :param x - float
21         :param y - int
22         :return x * y
23     """
24     if y == 0:
25         return 0
26     return add(x, mult(x, int(subtract_1(y))))
27
28
29 def is_even(n: int) -> bool:
30     """
31         Checks if n is even
32         :param n - int
33         :return is even?
34     """
35     if n == 1:
36         return False
37     elif n == 0:
38         return True
39     return is_even(subtract_1(subtract_1(n)))
40
41
42 def log_mult(x: float, y: int) -> float:
43     """
44         x * y, but O(log(n))
45         :param x - float
46         :param y - int
47         :return x * y
48     """
49     if y == 0:
50         return 0
51     if is_odd(y):
52         return add(log_mult(x, int(add(y, -1))), x)
53     else:
54         return add(log_mult(x, divide_by_2(y)), log_mult(x, divide_by_2(y)))
55
56
57 def is_power(b: int, x: int) -> bool:
58     """
59         checks if b is power of x:
```

```
60          :param b - int
61          :param x - int
62          :return is power?
63      """
64      if x == 1:
65          return True
66      elif b == 0 and x > 1:
67          return False
68      elif b == 0 and (x == 0 or x == 1):
69          return True
70      elif b > x:
71          return False
72
73      return is_power_helper(b, x, b, b)
74
75
76  def is_power_helper(b: int, x: int, c: int, new_b: int) -> bool:
77      """
78          checks if b is power of x:
79          :param b - int
80          :param x - int
81          :param c = b
82          :param new_b = b
83          :return is power?
84      """
85      if x == b:
86          return True
87      elif b > x:
88          return False
89      if b == add(x, 1):
90          return False
91      else:
92          power = int(log_mult(b,new_b))
93          is_power_helper(power, x, c, power)
94          b = int(log_mult(b, c))
95          return is_power_helper(b, x, c, c) or False
96
97
98  def reverse(s: str) -> str:
99      """
100         reverses a string
101         :param s - string
102         :return a reversed string
103     """
104     return reverse_helper(s, len(s), '',)
105
106
107 def reverse_helper(s: str, n: int, new_s: str) -> str:
108     """
109         helper funtion to reverse
110         :param s - string
111         :param n - len of s
112         :param new_s - the new string
113         :return revered string s
114     """
115     if n == 0:
116         return new_s
117     else:
118         return reverse_helper(s, n-1, append_to_end(new_s, s[n-1]))
119
120
121 # --------- Part 2 ---------
122
123
124 def play_hanoi(hanoi: Any, n: int, src: Any, dst: Any, temp: Any) -> None:
125     """
126         Solves the tower of Hanoi. Gets objects from hanoi_game.py
127         :param hanoi - float
```

```
        :param n - int
        :param src - Any
        :param dst - Any
        :param temp - Any
    """
    if n <= 0:
        return
    if n == 1:
        hanoi.move(src, dst)
    else:
        play_hanoi(hanoi, n-1, src, temp, dst)
        hanoi.move(src, dst)
        play_hanoi(hanoi, n-1, temp, dst, src)


# --------- Part 3 ---------


def number_of_ones(n:int) -> int:
    """
        conts the number of ones from 1 to n
        :param n - int
        :return number of ones
    """
    if n == 0:
        return 0
    else:
        return number_of_ones(n-1) + count_ones(n)


def count_ones(n: int) -> int:
    """
        counts the number of ones in n
        :param n - int
        :return number of ones in n
    """
    if n == 0:
        return 0
    else:
        if n%10 == 1:
            return number_of_ones(n//10) + 1
        else:
            return number_of_ones(n//10)


def compare_2d_lists(l1: List[List[int]], l2: List[List[int]]) -> bool:
    """
        deep compare 2D lists
        :param l1 - 2D list
        :param l2 - 2D list
        :return are they equal?
    """
    if len(l1) != len(l2):
        return False
    else:
        return compare_2d_lists_helper(l1, l2, len(l1))


def compare_2d_lists_helper(l1: List[List[int]],
                            l2: List[List[int]],
                            index: int) -> bool:
    """
        an helper function
        :param l1 - 2D list
        :param l2 - 2D list
        :return are they equal?
    """
    if index == 0:
```

```
196             return True
197         else:
198             return compare_2d_lists_helper(l1, l2, index-1) \
199                    and len(l1[index-1]) == len(l2[index-1]) \
200                    and compare_1d_lists(l1[index-1], l2[index-1], index)
201
202
203     def compare_1d_lists(l1: List[int], l2: List[int], index: int) -> bool:
204         """
205             Compare 1D list
206             :param l1 - 1D list
207             :param l2 - 1D list
208             :return are they equal?
209         """
210         if index == -1:
211             return True
212         else:
213             if len(l1) == 0 and len(l2) == 0:
214                 return True
215             return compare_1d_lists(l1, l2, index-1)\
216                 and l1[index-1] == l2[index-1]
217
218
219     def magic_list(n: int) -> List[Any]:
220         """
221             Makes a "magic" list
222             :param n - the depth of the list
223             :return a magic list
224         """
225         if n == 0:
226             return []
227         elif n == 1:
228             return [[]]
229         return magic_list_helper(n-1, [])
230
231
232     def magic_list_helper(n: int, lst: List[Any]) -> List[Any]:
233         """
234             an helper function
235             :param n - the depth of the list
236             :param lst - an empty list
237             :return a magic lst
238         """
239         if n == 0:
240             lst.append([])
241             return lst
242         else:
243             magic_list_helper(n-1, lst)
244             lst.append([])
245             magic_list_helper(n-1, lst[n])
246             return lst
```