

# Contents

1	Basic Test Results	2
2	aaa expected autograde	4
3	aaa hint result.png	5
4	AlignDNA.py	6
5	GetToTheZero.py	8
6	NonRecursiveMystery.py	10
7	README	11

# 1 Basic Test Results

```
1 Starting tests...
2 Wed Nov 27 14:13:42 IST 2013
3 9593320a69f98686648306fcd8ba588205de51d4 -
4
5
6 README
7 NonRecursiveMystery.py
8 GetToTheZero.py
9 AlignDNA.py
10
11 Testing README...
12 Done testing README...
13
14 Testing NonRecursiveMystery.py...
15 result_code    mystery    650    1
16 Done testing NonRecursiveMystery.py
17
18 Testing GetToTheZero.py...
19 result_code    getzero    143    1
20 result_code    getOverflow 10    1
21 Done testing GetToTheZero.py
22
23 Testing AlignDNA.py...
24 Timeout limit was 4 seconds
25 result_code    getbonus_0  timeout  1
26 Timeout limit was 4 seconds
27 result_code    getbonus_1  timeout  1
28 Timeout limit was 4 seconds
29 result_code    getbonus_2  timeout  1
30 Timeout limit was 4 seconds
31 result_code    getbonus_3  timeout  1
32 Timeout limit was 4 seconds
33 result_code    getbonus_4  timeout  1
34 result_code    getbonus    0    1
35 result_code    getbest     53    1
36 result_code    getscore    142    1
37 Done testing AlignDNA.py
38
39 Grading summary
40 -----
41 ***** mystery:
42 Number of failed tests: 0
43 Total number of tests : 650
44 Penalty: 0.0
45 ***** getzero:
46 Number of failed tests: 0
47 Total number of tests : 143
48 Penalty: 0.0
49 ***** getOverflow:
50 Number of failed tests: 0
51 Total number of tests : 10
52 Penalty: 0.0
53 ***** getbest:
54 Number of failed tests: 0
55 Total number of tests : 53
56 Penalty: 0.0
57 ***** getscore:
58 Number of failed tests: 0
59 Total number of tests : 142
```

```
60 Penalty: 0.0
61 ***** getbonus:
62 Number of passed tests: 0
63 Total number of tests : 5
64 Bonus: 0.0
65 *****
66 Expected automatic grade: 100.0
67 *****
68 Submission passed!
69 Tests completed
```

## 2 aaa expected autograde

```
1  Grading summary
2  -----
3  ***** mystery:
4  Number of failed tests: 0
5  Total number of tests : 650
6  Penalty: 0.0
7  ***** getzero:
8  Number of failed tests: 0
9  Total number of tests : 143
10 Penalty: 0.0
11 ***** get0overflow:
12 Number of failed tests: 0
13 Total number of tests : 10
14 Penalty: 0.0
15 ***** getbest:
16 Number of failed tests: 0
17 Total number of tests : 53
18 Penalty: 0.0
19 ***** getscore:
20 Number of failed tests: 0
21 Total number of tests : 142
22 Penalty: 0.0
23 ***** getbonus:
24 Number of passed tests: 0
25 Total number of tests : 5
26 Bonus: 0.0
27 *****
28 Expected automatic grade: 100.0
29 *****
30 Submission passed!
```

**3 aaa hint result.png**



## 4 AlignDNA.py

```
1 #####
2 # FILE: AlingDNA.py
3 # WRITER: Roi Greenberg + roigreenberg + 305571234
4 # EXERCISE : intro2cs ex5 2013-2014
5 # Description: calculate the match between 2 DNA strands
6 # and find the best match between two unaligned DNA strands
7 #####
8
9
10
11 def get_alignment_score(dna1,dna2,match=1,mismatch=-1,gap=-2):
12     ''' Calculate the match between 2 DNA strands
13
14     A function that calculate the match between 2 given DNA strands
15     according to 3 parameters. 'match' for the same DNA, 'mismatch'
16     for different letter and 'gap' for 1 letter and gap.
17
18     Args:
19     - dna1: first DNA strand. combined with the letters 'A' 'T' 'G' 'C'
20       and '-'.
21     - dna2: second DNA strand. combined with the letters 'A' 'T' 'G' 'C'
22       and '-'.
23     - match: score for the same DNA. integer number. default 1.
24     - mismatch: score for different letter. integer number. default -1.
25     - gap: score for for 1 letter and gap. integer number. default -2.
26
27     return: the score of the match'''
28
29     score=0 # start value
30     GAP="-"
31
32     # run for the length of the strands
33     for dna in range(len(dna1)):
34         # check for match
35         if dna1[dna]==dna2[dna]:
36             score += match
37         # check for mismatch and gap
38         elif dna1[dna]!=dna2[dna]:
39             if dna1[dna]!=GAP and dna2[dna]!="-":
40                 score += mismatch
41             else:
42                 score += gap
43     return score
44
45 def get_best_alignment_score(dna_1,dna_2,match=1,mismatch=-1,gap=-2):
46     ''' Find the match between two unaligned DNA strands
47
48     A function that find the best match between 2 given DNA strands
49     according to 3 parameters. 'match' for the same DNA, 'mismatch'
50     for different letter and 'gap' for 1 letter and gap.
51
52     Args:
53     - dna_1: first DNA strand. combined with the letters 'A' 'T' 'G' 'C'
54       and '-'.
55     - dna_2: second DNA strand. combined with the letters 'A' 'T' 'G' 'C'
56       and '-'.
57     - match: score for the same DNA. integer number. default 1.
58     - mismatch: score for different letter. integer number. default -1.
59     - gap: score for for 1 letter and gap. integer number. default -2.
```

Global constants should be declared in global scope, not function scope.

Why aren't you using GAP here? -1

the 'elif' in line 38 is unnecessary. If they aren't equal, then of course they are not equal. It should be exactly the same, but with an if...elif... else structure an the conditions in lines 35 and 39.

```

60
61     return: list with the score of the best match and both DNA strands'''
62
63     # define a list for all the matches
64     store=[]
65
66     def find_matches(dna_1_src,dna_2_src,dna_1_new="",dna_2_new=""):
67         ''' Find all matches between two unaligned DNA strands
68
69         A function that find all matches between 2 given DNA strands
70
71         Args:
72         - dna_1_src: source of the first DNA strand. combined with the letters
73           'A' 'T' 'G' 'C' and '-'.
74         - dna_2_src: source of the second DNA strand. combined with the letters
75           'A' 'T' 'G' 'C' and '-'.
76         - dna_1_new: new list build from the first source strands. start with
77           empty sequence
78         - dna_2_new: new list build from the second source strands. start with
79           empty sequence
80         return: list with lists of every matches'''
81
82         GAP="-"
83
84         # if no DNA left on source strands add the strand
85         if len(dna_1_src)==0 and len(dna_2_src)==0:
86             store.append([dna_1_new,dna_2_new])
87
88         # add first current letters from both strands
89         # if DNA left on source strands
90         if len(dna_1_src)!=0 and len(dna_2_src)!=0:
91             find_matches(dna_1_src[1:],dna_2_src[1:],\
92                           dna_1_new+dna_1_src[0],dna_2_new+dna_2_src[0])
93
94         # add first current letter from first strand and gap to the second
95         # if DNA left on first source strand
96         if len(dna_1_src)!=0:
97             find_matches(dna_1_src[1:],dna_2_src,\
98                           dna_1_new+dna_1_src[0],dna_2_new+GAP)
99
100        # add first current letter from second strand and gap to the first
101        # if DNA left on second source strand
102        if len(dna_2_src)!=0:
103            find_matches(dna_1_src,dna_2_src[1:],\
104                          dna_1_new+GAP,dna_2_new+dna_2_src[0])
105
106        # call the recursive function
107        find_matches(dna_1,dna_2)
108
109        # calculate the score for each dna match
110        for dna in store:
111            dna1=dna[0]
112            dna2=dna[1]
113            dna.insert(0,get_alignment_score(dna1,dna2,match,mismatch,gap))
114
115        # return the best match
116        return max(store)
117
118
119
120
121
122
123
124

```

The only reason to declare a function inside another function is if you are returning the inner function. Otherwise, declare it outside.  
-1

You're kind of missing the point of the recursion. You're supposed to use the result of a smaller problem, i.e. a recursive call, to construct the answer to a bigger problem. Not construct all the possible results step by step and then finding the best one.  
-2

## 5 GetToTheZero.py

```
1 #####
2 # FILE: GetToTheZero.py
3 # WRITER: Roi Greenberg + roigreenberg + 305571234
4 # EXERCISE : intro2cs ex5 2013-2014
5 # Description: Check if James Bond can pass the corridor undetected
6 #####
7
8 def is_solvable(start,corridor):
9     '''return a function that Check if James Bond can pass the corridor
10
11     Args:
12     - start: the start position. non negative integer smaller than corridor
13         length
14     - corridor: A list with positive integer except for the last place
15         that hold zero.
16
17     return: True if solvable, else False
18
19     In case of bad input: values are out of range
20     returns False'''
21
22     ## verifies the position is in the corridor
23     if start not in range(len(corridor)):
24         return False
25
26     END=0
27     ## create empty list for used positions
28     used_position=[]
29     def get_to_zero(position,corridor):
30         '''Check if James Bond can pass the corridor undetected
31
32         A function that check if a given corridor is possible to solve by
33         moving forward and backward according to the number in the current
34         position in order to get to the last position that hold 'zero'.
35
36         Args:
37         - start: the start position. non negative integer smaller than corridor
38             length
39         - corridor: A list with positive integer except for the last place
40             that hold zero.
41
42         return: True if solvable, else False
43
44         In case of bad input: values are out of range
45         returns False'''
46
47         # define the value of the finish value
48         END=0
49
50         # check if position already checked
51         if position in used_position:
52             return False
53
54         # add position to used positions list
55         used_position.append(position)
56
57         # check if the position is at the finish position
58         if corridor[position]==END:
59             return True
```

Again, helper  
functions should be  
declared outside.



```

60
61     # search the corridor forward and backward(if possible)
62     return (False if position+corridor[position]>=len(corridor) \
63             else get_to_zero(position+corridor[position],corridor)) \
64             or \
65             (False if position-corridor[position]<0 else\
66              get_to_zero(position-corridor[position],corridor))
67
68     return get_to_zero(start,corridor)

```

## 6 NonRecursiveMystery.py

```
1 #####
2 # FILE: NonRecursiveMystery.py
3 # WRITER: Roi Greenberg + roigreenberg + 305571234
4 # EXERCISE : intro2cs ex5 2013-2014
5 # Description: Implement a recursive function for sum of
6 #               divisors in iterative way
7 #####
8
9
10 def mystery_computation(number):
11     '''Implement a recursive function for sum of divisors in iterative way
12
13     A function that sum the divisors of a given number
14     (divisors=without remainder)
15
16     Args:
17     - number: integer number
18
19     return: sum of all the number divisors without reminders'''
20
21     # equal the sum to zero
22     divisors_sum=0
23
24     # run for all possible divisors
25     for divisor in range(1,number//2+1):
26         # if no reminders, add to sum
27         if number%divisor==0:
28             divisors_sum+=divisor
29
30     return divisors_sum
```

## 7 README

```
1  roigreenberg
2  305571234
3  roi greenberg
4
5  =====
6  =  README for ex5: Recursion  =
7  =====
8
9
10
11 =====
12 =  Description:  =
13 =====
14
15 NonRecursiveMystery: Implement a recursive function for sum of divisors in iterative way
16                      (divisors=without remainder).
17 GetToTheZero: Check if James Bond can pass the corridor undetected.
18 AlignDNA: Task 1: Calculate the match between 2 DNA strands with 3 optimal score for match, mismatch and gap.
19           Task 2: Find the match between two unaligned DNA strands that give the best score
20                 as calculate in task 1
21
22 =====
23 =  List of submitted files:  =
24 =====
25
26 README                This file
27 NonRecursiveMystery.py    Iterative way for recursive function for sum of divisors
28 GetToTheZero.py          Check if James Bond can pass the corridor undetected
29 AlignDNA:                calculate the match between 2 DNA strands
30                           and find the best match between two unaligned DNA strands
31
32 =====
33 =  Special Comments  =
34 =====
```