

# Contents

<b>1</b>	<b>Basic Test Results</b>	<b>2</b>
<b>2</b>	<b>ex3.py</b>	<b>4</b>

# 1 Basic Test Results

```
1 Sat Nov 14 23:37:41 IST 2020
2 Process Process-78:
3 Traceback (most recent call last):
4   File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
5     self.run()
6   File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
7     self._target(*self._args, **self._kwargs)
8   File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/autotest.py", line 74, in wrap
9     res=target(*args, **kwargs)
10  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 39, in import_runner
11    code,res = peel(runners, modulename, fname, args, kwargs)
12  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 7, in peel
13    return runners[-1](modulename, fname, args, kwargs,options,runners[:-1])
14  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 17, in check_args
15    code,res = peel(runners, modulename, fname, args, kwargs)
16  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 7, in peel
17    return runners[-1](modulename, fname, args, kwargs,options,runners[:-1])
18  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 12, in base_runner
19    return None,func(*args, **kwargs)
20  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/src/ex3.py", line 184, in primes_for_asafi
21    if is_prime(list(num_range)[i]):
22  IndexError: list index out of range
23 Process Process-82:
24 Traceback (most recent call last):
25   File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
26     self.run()
27   File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
28     self._target(*self._args, **self._kwargs)
29   File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/autotest.py", line 74, in wrap
30     res=target(*args, **kwargs)
31  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 39, in import_runner
32    code,res = peel(runners, modulename, fname, args, kwargs)
33  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 7, in peel
34    return runners[-1](modulename, fname, args, kwargs,options,runners[:-1])
35  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 17, in check_args
36    code,res = peel(runners, modulename, fname, args, kwargs)
37  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 7, in peel
38    return runners[-1](modulename, fname, args, kwargs,options,runners[:-1])
39  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/lib/testrunners.py", line 12, in base_runner
40    return None,func(*args, **kwargs)
41  File "/tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/testdir/src/ex3.py", line 191, in sum_of_vectors
42    vector_length = len(vec_lst[0])
43  IndexError: list index out of range
44 Sat Nov 14 23:37:41 IST 2020
45 Archive: /tmp/bodek.KRNTQq/intro2cs1/ex3/tsviel/final/submission
46   inflating: src/ex3.py
47   inflating: src/_MACOSX/._ex3.py
48 7 passed tests out of 7 in test set named 'presubmit'.
49 result_code  presubmit    7    1
50 12 passed tests out of 12 in test set named 'input'.
51 result_code   input     12    1
52 14 passed tests out of 14 in test set named 'inner'.
53 result_code   inner     14    1
54 14 passed tests out of 14 in test set named 'monotonicity'.
55 result_code  monotonicity 14    1
56 22 passed tests out of 22 in test set named 'inverse'.
57 result_code   inverse    22    1
58 --> BEGIN TEST INFORMATION
59 Test name: prime_t8
```

```

60 Module tested: ex3
61 Function call: primes_for_asafi(5000)
62 Expected return value: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101,
63 More test options: {}
64 --> END TEST INFORMATION
65 *****
66 *****          There is a problem:
67 *****          The test named 'prime_t8' failed.
68 *****
69 Test did not complete, exited with exitcode 1.
70 This probably means your code caused an exception to be raised.
71 result_code  prime_t8  exception  1
72 8 passed tests out of 9 in test set named 'prime'.
73 result_code  prime      8      1
74 --> BEGIN TEST INFORMATION
75 Test name: vecsums_t3
76 Module tested: ex3
77 Function call: sum_of_vectors([])
78 Expected return value: None
79 More test options: {}
80 --> END TEST INFORMATION
81 *****
82 *****          There is a problem:
83 *****          The test named 'vecsums_t3' failed.
84 *****
85 Test did not complete, exited with exitcode 1.
86 This probably means your code caused an exception to be raised.
87 result_code  vecsums_t3  exception  1
88 8 passed tests out of 9 in test set named 'vecsums'.
89 result_code  vecsums      8      1
90 11 passed tests out of 11 in test set named 'orthogonal'.
91 result_code  orthogonal  11      1
92 TESTING COMPLETED

```

## 2 ex3.py

```
1  STOP = ""
2  WAITING_FOR_INPUT = True
3  PASSIVE_PRODUCT = 1
4  PASSIVE_SUM = 0
5
6
7  def lst_sum(lst):
8      """The function returns the sum of a list of numbers"""
9      the_sum = 0
10     for i in range(len(lst)):
11         the_sum += lst[i]
12     return the_sum
13
14
15 def input_list():
16     """The Function recieves numbers from users input,
17     and returns them in a list from the first to the last one with there sum"""
18     lst = []
19
20     while WAITING_FOR_INPUT:
21         user_input = input()
22         if user_input == STOP:
23             break;
24         else:
25             lst.append(float(user_input))
26     return lst + [lst_sum(lst)]
27
28
29 def iter_product(iterable):
30     """
31     Returns the product of an Iterable
32     :param iterable: Any iterable dataset (set, tuple,
33     list, etc..) of floats\int
34     :return: The inner product of the iterable unit
35     """
36     output = PASSIVE_PRODUCT
37     for num in iterable:
38         output *= num
39     return output
40
41
42 def inner_product(vec_1, vec_2):
43     """
44     :param vec_1: a list of numbers(vector)
45     :param vec_2: a list of numbers(vector)
46     :return: A number(Float) represnting The inner product
47     """
48     # Validate if the abs of the vectors are equal
49     if len(vec_1) != len(vec_2):
50         return None
51     # Validate if that the lists are not empty
52     if len(vec_2) == 0 or len(vec_1) == 0:
53         return 0
54     # Action
55     output = []
56     for iterable in zip(vec_1, vec_2):
57         output.append(iter_product(iterable))
58     return lst_sum(output)
59
```

```

60
61 def monotonicity_up(sequence):
62     """
63     :param sequence: an Iterable of numbers(floats or ints)
64     :return: True if given sequence is Monotonicity up, False if not
65     """
66     cache = []
67     for item in zip(sequence[:], sequence[1:]):
68         if item[0] <= item[1]:
69             cache.append(True)
70         else:
71             cache.append(False)
72     return iter_product(cache)
73
74
75 def monotonicity_up_abs(sequence):
76     """
77     :param sequence: an Iterable of numbers(floats or ints)
78     :return: True if given sequence is Monotonicity very up, False if not
79     """
80     cache = []
81     for item in zip(sequence[:], sequence[1:]):
82         if item[0] < item[1]:
83             cache.append(True)
84         else:
85             cache.append(False)
86     return iter_product(cache)
87
88
89 def monotonicity_down(sequence):
90     """
91     :param sequence: an Iterable of numbers(floats or ints)
92     :return: True if given sequence is Monotonicity down, False if not
93     """
94     cache = []
95     for item in zip(sequence[:], sequence[1:]):
96         if item[0] >= item[1]:
97             cache.append(True)
98         else:
99             cache.append(False)
100     return iter_product(cache)
101
102
103 def monotonicity_down_abs(sequence):
104     """
105     :param sequence: an Iterable of numbers(floats or ints)
106     :return: True if given sequence is Monotonicity very down, False if not
107     """
108     cache = []
109     for item in zip(sequence[:], sequence[1:]):
110         if item[0] > item[1]:
111             cache.append(True)
112         else:
113             cache.append(False)
114     return iter_product(cache)
115
116
117 def sequence_monotonicity(sequence):
118     """
119     :param sequence: List of Integers
120     :return: List of Booleans
121     """
122     output = [] # The output to store our boolean answers
123     if not isinstance(sequence, type(None)):
124         if (sequence == [] or len(sequence) == 1):
125             return [True, True, True, True]
126         else:
127             if monotonicity_up(sequence): # Monotonicity Up Case

```

```

128         output.append(True)
129     else:
130         output.append(False)
131     if monotonicity_up_abs(sequence): # Monotonicity Very up Case
132         output.append(True)
133     else:
134         output.append(False)
135     if monotonicity_down(sequence): # Monotonicity Down Case
136         output.append(True)
137     else:
138         output.append(False)
139     if monotonicity_down_abs(sequence): # Monotonicity Very Down Case
140         output.append(True)
141     else:
142         output.append(False)
143     return output # Return answer for monotonicity
144
145
146
147
148 def monotonicity_inverse(def_bool):
149     if def_bool == [True, True, False, False]:
150         return [-4, -3, 2, 5] # Motonocity VERY UP
151     if def_bool == [True, False, False, False]:
152         return [1, 2, 2, 3] # MONOTONICITY UP
153     if def_bool == [False, False, True, True]:
154         return [4.2, 3, 0, -2] #MONOTONICITY DOWN
155     if def_bool == [False, False, True, False]:
156         return [4, 3, 3, 1] # MOTONICITY VERY DOWN
157     if def_bool == [True, True, True, True]:
158         return None
159     if def_bool == [True, False, True, False]:
160         return [1, 1, 1, 1]
161     if def_bool == [False, False, False, False]:
162         return [1, 2, -5, -1]
163     else:
164         return None
165
166
167 def is_prime(d):
168     """True if n is prime number, False if not"""
169     i = 2
170     while i < d:
171         if d % i == 0:
172             return False
173         i += 1
174     return True
175
176
177 def primes_for_asafi(n):
178     """ Help Asafi to return the prime numbers from 1 to n"""
179     num_range = range(1, 10000)
180     primes = []
181     for i in num_range:
182         if len(primes) == n:
183             break
184         if is_prime(list(num_range)[i]):
185             primes.append(list(num_range)[i])
186     return primes
187
188
189 def sum_of_vectors(vec_lst):
190     """Returns the sum of of vectors"""
191     vector_length = len(vec_lst[0])
192     sum_vector = []
193     for coordinate in range(vector_length):
194         sum_vector_coordinate = []
195         for vector_index in range(len(vec_lst)):

```

```

196         sum_vector_coordinate.append(vec_lst[vector_index][coordinate])
197     sum_vector.append(lst_sum(sum_vector_coordinate))
198     return sum_vector
199
200
201 def num_of_orthogonal(vectors):
202     """
203     :param vectors: A Matrix (2D List) of vectors of same length
204     :return: The amount of orthogonal Vectors
205     """
206     count = 0
207     for vector_a in range(len(vectors)):
208         for vector_b in range(vector_a+1, len(vectors)):
209             if inner_product(vectors[vector_a], vectors[vector_b]) == 0:
210                 count += 1
211     return count
212

```