

## Problem 0 Source

```
1 class Solution:
2     def is_palindrome(s):
3         i = 0
4         j = len(s) - 1
5         while i < j:
6             if s[i] != s[j]:
7                 return False
8             i += 1
9             j -= 1
10
11     return True
12
13 s = Solution()
14 print s.is_palindrome("foobar")
15 print s.is_palindrome("hannah")
```

## Problem 0 Output

```
1 Traceback (most recent call last):
2   File "Problem 0", line 14, in <module>
3     print s.is_palindrome("foobar")
4 TypeError: is_palindrome() takes exactly 1 argument (2 given)
```

## Problem 1 Source

```
1 class Solution:
2     def is_palindrome(self, s):
3         i = 0
4         j = len(s) - 1
5         while i < j:
6             if s[i] != s[j]:
7                 return False
8             i += 1
9             j -= 1
10
11     return True
12
13 print Solution.is_palindrome("foobar")
14 print Solution.is_palindrome("hannah")
15
```

## Problem 1 Output

```
1 Traceback (most recent call last):
2   File "Problem 1", line 13, in <module>
3     print Solution.is_palindrome("foobar")
4 TypeError: unbound method is_palindrome() must be called with
Solution instance as first argument (got str instance instead)
```

## Problem 2 Source

```
1 # looks for sub inside str
2 def find_substring(str, sub):
3     return str.find(sub)
```

```

4
5 print find_substring("finding fens")

```

## Problem 2 Output

```

1 Traceback (most recent call last):
2   File "Problem 2", line 5, in <module>
3     print find_substring("finding fens")
4   TypeError: find_substring() takes exactly 2 arguments (1 given)

```

## Problem 3 Source

```

1  # Given an unsorted array of integers, find the length of longest continuous
increasing subsequence (subarray).
2  # Input: [1,3,5,4,7]
3  # Output: 3
4
5
6  def lcis(arr):
7      i = 0
8      cur_count = 0
9      max_count = 0
10
11     while (i < len(arr)):
12         if arr[i] < arr[i+1]:
13             cur_count += 1
14             max_count = max(cur_count, max_count)
15         i += 1
16
17     return max_count
18
19 print(lcis([1,3,5,4,7]))

```

## Problem 3 Output

```

1 Traceback (most recent call last):
2   File "Problem 3", line 19, in <module>
3     print(lcis([1,3,5,4,7]))
4   File "Problem 3", line 12, in lcis
5     if arr[i] < arr[i+1]:
6   IndexError: list index out of range

```

## Problem 4 Source

```

1  def is_palindrome(s, i, j):
2      if s[i] != s[j]:
3          return False
4
5      return is_palindrome(s, i+1, j-1)
6
7  print is_palindrome("foobar", 0, len("foobar") - 1)
8  print is_palindrome("hannah", 0, len("hannah") - 1)

```

## Problem 4 Output

```
1 False
2 Traceback (most recent call last):
3   File "Problem 4", line 8, in <module>
4     print is_palindrome("hannah", 0, len("hannah") - 1)
5   File "Problem 4", line 5, in is_palindrome
6     return is_palindrome(s, i+1, j-1)
7   File "Problem 4", line 5, in is_palindrome
8     return is_palindrome(s, i+1, j-1)
9   File "Problem 4", line 5, in is_palindrome
10    return is_palindrome(s, i+1, j-1)
11   File "Problem 4", line 5, in is_palindrome
12    return is_palindrome(s, i+1, j-1)
13   File "Problem 4", line 5, in is_palindrome
14    return is_palindrome(s, i+1, j-1)
15   File "Problem 4", line 5, in is_palindrome
16    return is_palindrome(s, i+1, j-1)
17   File "Problem 4", line 2, in is_palindrome
18     if s[i] != s[j]:
19 IndexError: string index out of range
```

## Problem 5 Source

```
1 # generates all subsets from array
2 def all_subsets(arr, i=0, cur=[], output=[]):
3     if i >= len(arr):
4         output.append(cur)
5         return
6
7     all_subsets(arr, i+1, cur, output)
8
9     cur.append(arr[i])
10    all_subsets(arr, i+1, cur, output)
11    cur.pop()
12    return output
13
14 # should print the following lists
15 # [], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]
16 # but prints [], [], [], [], [], [], [], [] instead
17 print all_subsets([1,2,3])
```

## Problem 5 Output

```
1 [], [], [], [], [], [], [], []
```

## Problem 6 Source

```
1 import sys
2 sys.setrecursionlimit(5)
3
4 def sum(arr, i=0):
5     return sum(arr, i+1) + arr[i]
6
7 print sum([3, 5, 2])
```

## Problem 6 Output

```
1 Traceback (most recent call last):
2   File "Problem 6", line 7, in <module>
3     print sum([3, 5, 2])
4   File "Problem 6", line 5, in sum
5     return sum(arr, i+1) + arr[i]
6   File "Problem 6", line 5, in sum
7     return sum(arr, i+1) + arr[i]
8   File "Problem 6", line 5, in sum
9     return sum(arr, i+1) + arr[i]
10  File "Problem 6", line 5, in sum
11    return sum(arr, i+1) + arr[i]
12 RuntimeError: maximum recursion depth exceeded
```

## Problem 7 Source

```
1 # list all duplicate values in arr
2 def find_duplicates(arr):
3     dupes = set()
4     seen = {}
5     for a in arr:
6         if seen[a] == True:
7             dupes.add(a)
8         else:
9             seen[a] = True
10
11     return seen
12
13 print find_duplicates([1,2,3,2,4,5,1])
```

## Problem 7 Output

```
1 Traceback (most recent call last):
2   File "Problem 7", line 13, in <module>
3     print find_duplicates([1,2,3,2,4,5,1])
4   File "Problem 7", line 6, in find_duplicates
5     if seen[a] == True:
6   KeyError: 1
```

## Problem 8 Source

```
1 def prod(arr, i=0):
2     if i >= len(arr):
3         return 1
4
5     return arr[i] * prod(arr, i+1)
6
7 # if you run this file nothing happens, why?
```

## Problem 8 Output

## Problem 9 Source

```
1 def sum(arr):
2     s = 0
3     for i in xrange(arr):
4         s += arr[i]
5     return s
6
7 arr = [1,3,5,7,9]
8 print sum(arr)
```

## Problem 9 Output

```
1 Traceback (most recent call last):
2   File "Problem 9", line 8, in <module>
3     print sum(arr)
4   File "Problem 9", line 3, in sum
5     for i in xrange(arr):
6   TypeError: an integer is required
```

## Problem 10 Source

```
1 def is_palindrome(s, i, j):
2     if i >= j:
3         return True
4
5     if s[i] != s[j]:
6         return False
7
8     return is_palindrome(s, i+1, j-1)
9
10 print is_palindrome("foobar")
11 print is_palindrome("hannah")
12
```

## Problem 10 Output

```
1 Traceback (most recent call last):
2   File "Problem 10", line 10, in <module>
3     print is_palindrome("foobar")
4   TypeError: is_palindrome() takes exactly 3 arguments (1 given)
```

## Problem 11 Source

```
1 def find_max(arr):
2     maxval = arr[0]
3     for val in arr:
4         maxval = max(val, maxval)
5
6
7 print find_max([10, 3, 9, 8, 21])
```

## Problem 11 Output

```
1 None
```

## Problem 12 Source

```
1  # Implement the function strStr().
2  # strStr takes two parameters a main string (haystack) and a substring (needle)
3  # and returns the the first index of the match. If there is no match, the
function will return -1
4  # i.e if haystack = "foo bar bar" and needle = "bar"
5  # the function will return 4
6
7  # if the needle is an empty string, the haystack is returned
8
9
10 def strStr(haystack, needle):
11     if len(needle) == 0:
12         return haystack
13
14     for i in range(len(haystack) - len(needle)):
15         if haystack[i: len(needle)] == needle:
16             return i
17     return -1
18
19 print(strStr("endless need for needles", "needle")) # the function should
return 17
```

## Problem 12 Output

```
1  -1
```

## Problem 13 Source

```
1  # this takes in a string like "abc" and
2  # generates all strings created by inserting
3  # a space at every position in the string.
4  # in this case, all strings will be
5  # "a bc" and "ab c"
6  # for "abcd", all strings will be
7  # "a bcd", "ab cd", "abc d"
8  def generate_all_words(s):
9      ret = []
10     for i in xrange(1, len(s)-1):
11         c = str(s)
12         c[i] = " "
13         ret.append(c)
14
15     return ret
16
17 generate_all_words("abcd")
```

## Problem 13 Output

```
1  Traceback (most recent call last):
2    File "Problem 13", line 17, in <module>
3        generate_all_words("abcd")
4    File "Problem 13", line 12, in generate_all_words
5        c[i] = " "
6  TypeError: 'str' object does not support item assignment
```

## Problem 14 Source

```
1 # returns if string s has the string 'needle' in it
2 # i.e. has_needle("has needle") should return True
3 # has_needle("foobar") should return False
4 def has_needle(s):
5     return s.find(needle) != -1
6
7
8 print has_needle("foobar")
9 print has_needle("has a needle")
```

## Problem 14 Output

```
1 Traceback (most recent call last):
2   File "Problem 14", line 8, in <module>
3     print has_needle("foobar")
4   File "Problem 14", line 5, in has_needle
5     return s.find(needle) != -1
6 NameError: global name 'needle' is not defined
```