## Problem 0 Source

```
class Solution:
    def is_palindrome(s):
        i = 0
        j = len(s) - 1
        while i < j:
            if s[i] != s[j]:
                return False
            i += 1
            j -= 1

        return True

s = Solution()
print s.is_palindrome("foobar")
print s.is_palindrome("hannah")
```

## Problem 0 Output

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

## Problem 1 Source

```
class Solution:
    def is_palindrome(self, s):
        i = 0
        j = len(s) - 1
        while i < j:
            if s[i] != s[j]:
                return False
            i += 1
            j -= 1

        return True

print Solution.is_palindrome("foobar")
print Solution.is_palindrome("hannah")
```

## Problem 1 Output

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

## Problem 2 Source

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

print find_substring("finding fens")
```

## Problem 2 Output

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

## Problem 3 Source

```
def is_palindrome(s, i, j):
    if s[i] != s[j]:
        return False

    return is_palindrome(s, i+1, j-1)

print is_palindrome("foobar", 0, len("foobar") - 1)
print is_palindrome("hannah", 0, len("hannah") - 1)
```

## Problem 3 Output

```
False
Traceback (most recent call last):
  File "Problem 3", line 8, in <module>
    print is_palindrome("hannah", 0, len("hannah") - 1)
  File "Problem 3", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 3", line 5, in is_palindrome
```

```
    return is_palindrome(s, i+1, j-1)
  File "Problem 3", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 3", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 3", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 3", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 3", line 2, in is_palindrome
    if s[i] != s[j]:
IndexError: string index out of range
```

## Problem 4 Source

```
# generates all subsets from array
def all_subsets(arr, i=0, cur=[], output=[]):
    if i >= len(arr):
        output.append(cur)
        return

    all_subsets(arr, i+1, cur, output)

    cur.append(arr[i])
    all_subsets(arr, i+1, cur, output)
    cur.pop()
    return output

# should print the following lists
# [], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]
# but prints [[], [], [], [], [], [], [], []] instead
print all_subsets([1,2,3])
```

## Problem 4 Output

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

## Problem 5 Source

```
import sys
sys.setrecursionlimit(5)

def sum(arr, i=0):
```

```
    return sum(arr, i+1) + arr[i]

print sum([3, 5, 2])
```

## Problem 5 Output

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

## Problem 6 Source

```
# list all duplicate values in arr
def find_duplicates(arr):
    dupes = set()
    seen = {}
    for a in arr:
        if seen[a] == True:
            dupes.add(a)
        else:
            seen[a] = True

    return seen

print find_duplicates([1,2,3,2,4,5,1])
```

## Problem 6 Output

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

## Problem 7 Source

```
def prod(arr, i=0):
    if i >= len(arr):
        return 1

    return arr[i] * prod(arr, i+1)

# if you run this file nothing happens, why?
```

## Problem 7 Output

## Problem 8 Source

```
def sum(arr):
    s = 0
    for i in xrange(arr):
        s += arr[i]
    return s

arr = [1,3,5,7,9]
print sum(arr)
```

## Problem 8 Output

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

## Problem 9 Source

```
def is_palindrome(s, i, j):
    if i >= j:
        return True

    if s[i] != s[j]:
        return False
```

```
    return is_palindrome(s, i+1, j-1)

print is_palindrome("foobar")
print is_palindrome("hannah")
```

### Problem 9 Output

```
Traceback (most recent call last):
  File "Problem 9", line 10, in <module>
    print is_palindrome("foobar")
TypeError: is_palindrome() takes exactly 3 arguments (1 given)
```

### Problem 10 Source

```
def find_max(arr):
    maxval = arr[0]
    for val in arr:
        maxval = max(val, maxval)


print find_max([10, 3, 9, 8, 21])
```

### Problem 10 Output

```
None
```

### Problem 11 Source

```
# Implement the function strStr().
# strStr takes two parameters a main string (haystack) and a substring (needle)
# and returns the the first index of the match. If there is no match, the function will retu
# i.e if haystack = "foo bar bar" and needle = "bar"
# the function will return 4

# if the needle is an empty string, the haystack is returned


def strStr(haystack, needle):
    if len(needle) == 0:
        return haystack
```

```
    for i in range(len(haystack) - len(needle)):
        if haystack[i: len(needle)] == needle:
            return i
    return -1

print(strStr("endless need for needles", "needle")) # the function should return 17
```

## Problem 11 Output

```
-1
```

## Problem 12 Source

```
# this takes in a string like "abc" and
# generates all strings created by inserting
# a space at every position in the string.
# in this case, all strings will be
# "a bc" and "ab c"
# for "abcd", all strings will be
# "a bcd", "ab cd", "abc d"
def generate_all_words(s):
    ret = []
    for i in xrange(1, len(s)-1):
        c = str(s)
        c[i] = " "
        ret.append(c)

    return ret

generate_all_words("abcd")
```

## Problem 12 Output

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

### Problem 13 Source

```
# returns if string s has the string 'needle' in it
# i.e. has_needle("has needle") should return True
# has_needle("foobar") should return False
def has_needle(s):
    return s.find(needle) != -1


print has_needle("foobar")
print has_needle("has a needle")
```

### Problem 13 Output

```
Traceback (most recent call last):
  File "Problem 13", line 8, in <module>
    print has_needle("foobar")
  File "Problem 13", line 5, in has_needle
    return s.find(needle) != -1
NameError: global name 'needle' is not defined
```